# US106-TRIACMTDEMOZ

The US106 TRIAC Reference Design Board

## Introduction

The purpose of this document is to detail the software that is used with the US106 TRIAC board. This software runs on the MCB-RX26T-TYPE-A.

## Contents

# 1. TRIAC Theory

The TRIAC is a 3-terminal device that allows the bidirectional flow of current. The terminals are traditionally referred to as MT1, MT2, and gate. Applying a sufficient current to gate enables current flow between MT1 and MT2 with the direction depending on their relative potentials. This flow of current continues until it is too small in magnitude causing the device not to conduct and requiring the gate to be retriggered.

This device can be used for a variety of applications, including motor speed or lighting control. By triggering the gate of the TRIAC at a specific time in the AC sine wave from the wall, the amount of the wave that is seen by the device can be controlled, allowing a motor to be sped up or a light to be dimmed.

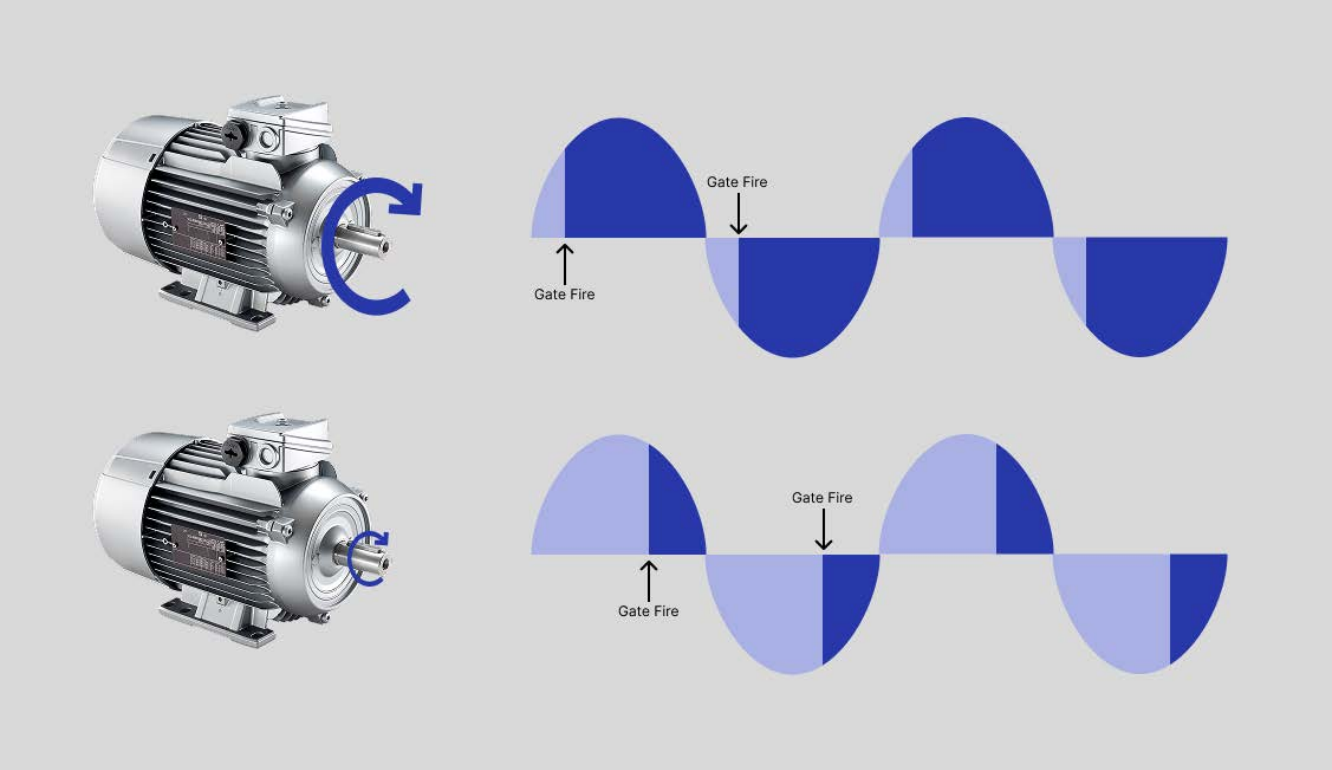



**Figure 1. Diagram of How a Firing Signal Works on a Sine Wave for a TRIAC Motor Drive**

# 2. Software Requirements

At its core, this application simply controls the timing of gate pulses to deliver the required amount of power to the output device. However, there a variety of other things the software must do to facilitate the safe and predictable behavior. The following list is comprehensive:

- Monitor the line voltage for zero crossings.
- Detect whether main frequency is 50Hz or 60Hz.
- Pulse the watchdog relay to keep the circuit active.
- Periodically fire the gate of three separate TRIACs to achieve their independently defined duty cycles.
- Monitor and report the angular velocity of a hall effect tachometer.
- Monitor and report the value of an onboard potentiometer.

# 3. Software Overview

This section is a general overview of the software flow.

## 3.1 Software Flow

The program can be thought of as having three distinct phases – initialization, normal operation, and exit. On program entry, the software is in the initialization phase. Here, all pins, ports, and peripherals are created and configured for use in the program. It also enforces a delay of approximately ten seconds before triggering the relay that is required for its operation. After the relay is triggered and power from the wall is delivered to the rest of the system, the zero-voltage detection (ZVC) circuit begins detecting the zero crossings of the AC waveform. The time between the first two crossings is measured with a timer to determine whether the input waveform is operating at 50Hz or 60Hz.

Following this calculation, the program moves into the normal operation phase so that when a ZVC is detected, each individual TRIAC is fired according to its respective duty cycle, which is currently set with a visual expression dial. Also in the background, interrupts are driving (1) the reading of voltage and current across the motors, (2) the rpm of an actuator from a hall effect sensor, and (3) the position of an onboard potentiometer. This continues until a HALT is requested, which moves the software into the exit phase. As a result, the watchdog relay trigger signal is stopped, and the program exits.

## 3.2 Block Diagrams

### 3.2.1 Main Diagram

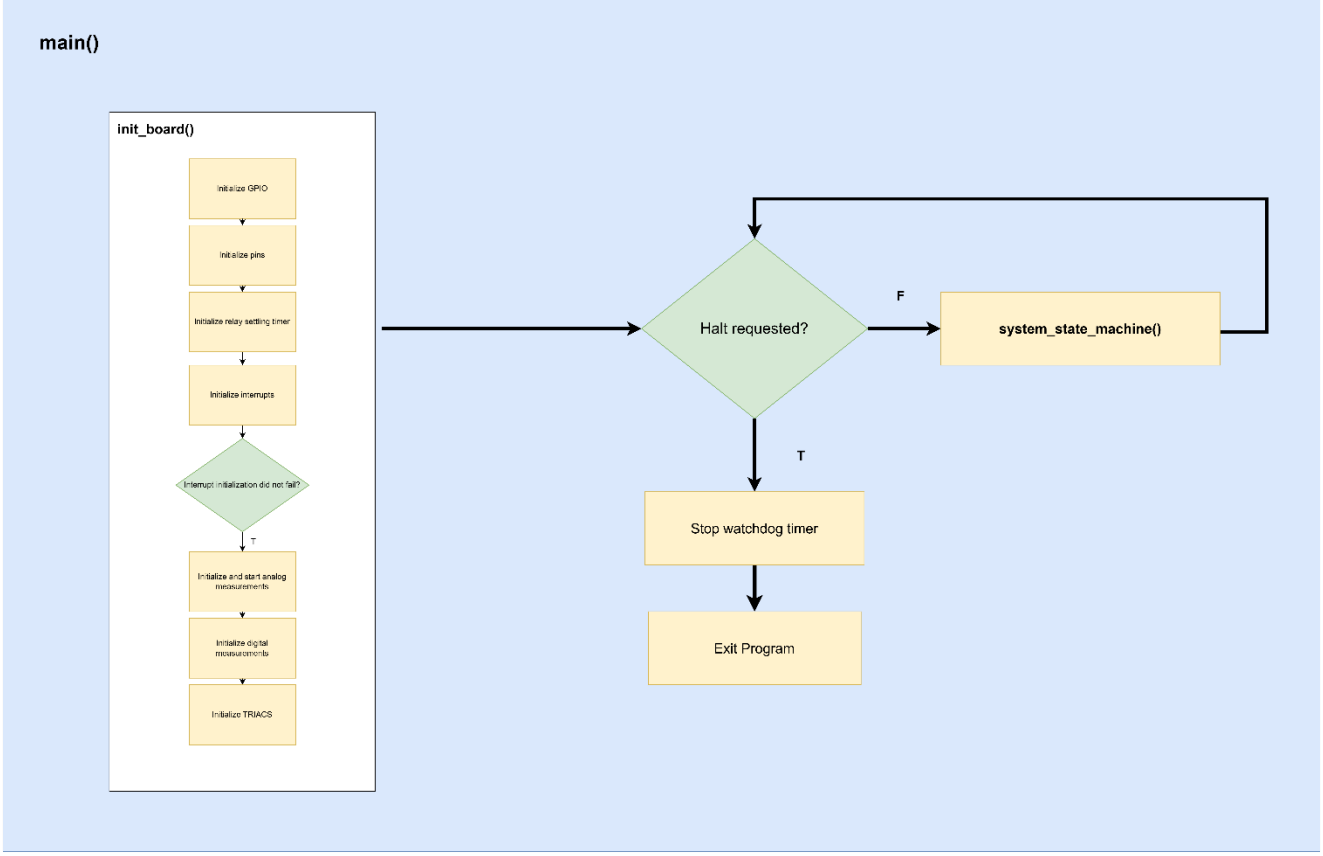



**Figure 2. Main Software Flow Diagram**

### 3.2.2 State Machine Block Diagram

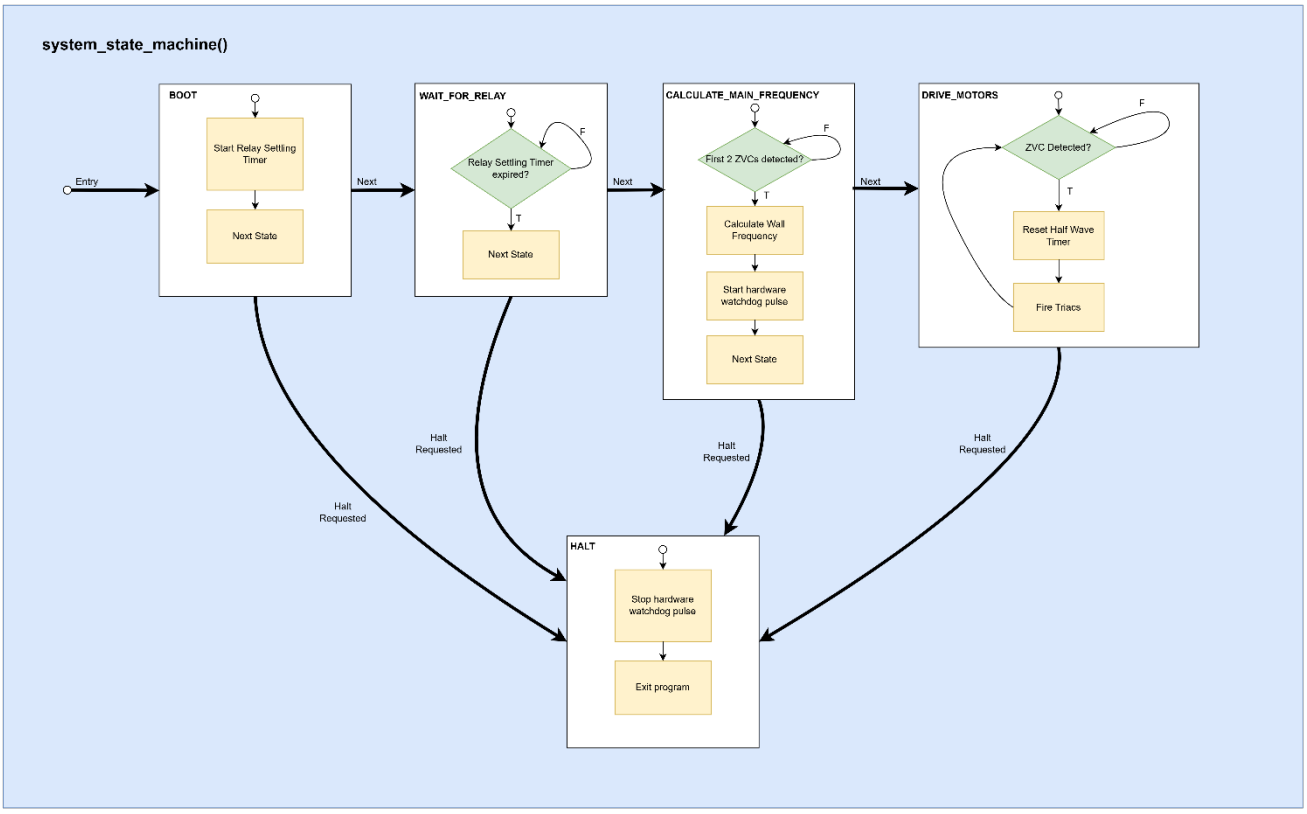
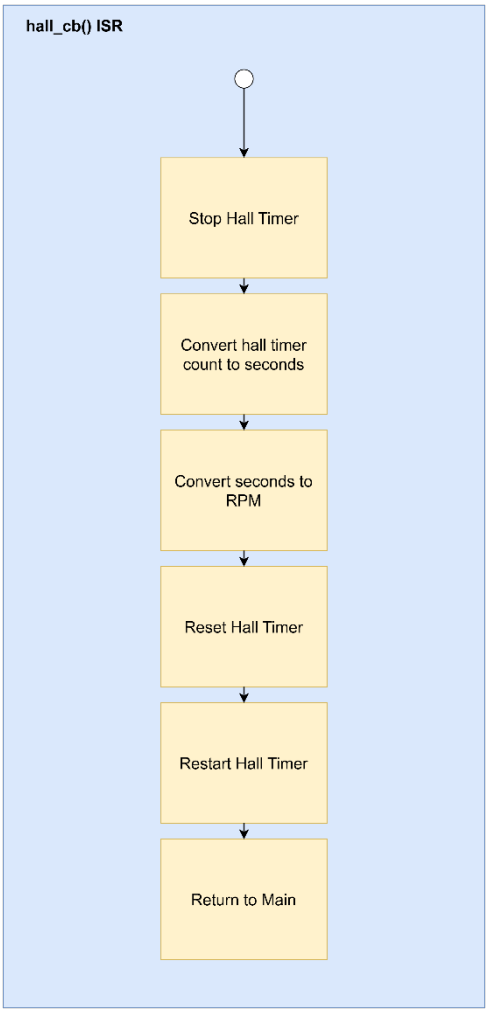



**Figure 3: Individual Depiction of the State Machines Controlling Each Software Function**

### 3.2.3 ISR Block Diagrams



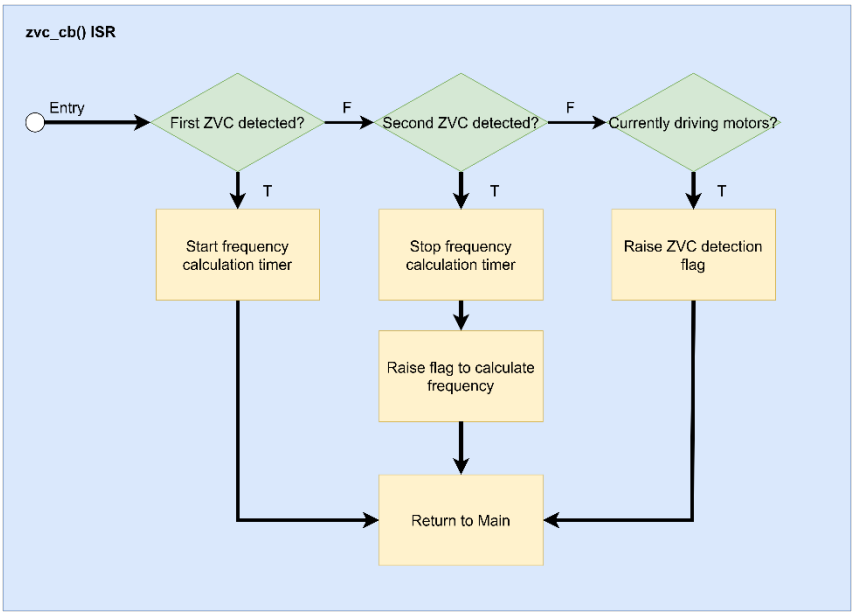




**Figure 4. Interrupt Service Routing Information Flow**

## 3.3 Data

This is a description of the custom types created for this application, and global variables that are shared across the program.

### 3.3.1 Enumerations

#### 3.3.1.1 triac_identifier_t

The triac_identifier_t data type exists to identify a specific TRIAC. This board has three TRIACs on it, so there are three fields. If this software is modified to support more TRIACs, the expectation is that triac_identifier_t would be augmented to include more unique identifiers. The global TRIAC array also uses this enumeration for indexing, so it is imperative that they are added to that array in the same order that they are defined in this enumeration. This enumeration is defined in Triac.h.

**Table 1. Enumerations for the TRIACs**

| Name | Integer | Meaning |
|---|---|---|
| MED | 0 | Medium current TRIAC |
| HP1 | 1 | High current TRIAC #1 |
| HP2 | 2 | High current TRIAC #2 |

#### 3.3.1.2 system_state

This data type defines the possible states for the system state machine. It is defined in System.h.

**Table 2. Enumerations for the System State**

| Name | Integer | Meaning |
|---|---|---|
| BOOT | 0 | Initial system state |
| WAIT_FOR_RELAY | 1 | Wait for delay timer to expire before beginning the pulsing gate of the watchdog relay timer |
| CALCULATE_MAIN_FREQUENCY | 2 | Wait for the first two ZVCs to be detected before calculating the frequency of AC Main |
| DRIVE_MOTORS | 3 | Core operation. Waiting for ZVC detection flag to pulse TRIAC gates |
| HALT | 4 | System halting operation |

### 3.3.2 Structures

#### 3.3.2.1 system_t

The system_t type holds data relevant to the entire system. This is a broad classification, but the structure holds information that is related to the program flow, state advancement, information about main's frequency, and timing constants that must be calculated on program startup. This data type is defined in System.h.

**Table 3. Structure Description and Definition for the System Variables**

| Field Type | Name | Description |
|---|---|---|
| volatile system_sm_state_t | state | The current state of the system state machine, which directs high level program flow |
| volatile bool | relay_delay_complete | Indicates whether the timer for delaying the start of safety relay operation has expired |
| volatile bool | two_zvcs_detected | Indicates whether the first two zero-voltage crossings have been detected, indicating that that the system has seen enough information to calculate the frequency of AC Main. |
| volatile bool | halt_requested | Indicates if a system halt has been requested. This flag is detected on the next state machine loop and begins the exit of the program |
| volatile bool | zvc_detected | Indicates whether a ZVC has been detected. This is ONLY set when the system state machine is actively driving motors. In that state, the state machine polls zvc_detected to see if it is time to fire the TRIACs. When it fires the TRIACs, it resets this field. |
| volatile bool | is_sixty_hertz | Indicates whether the frequency of AC Main is 60hz. If false, the line is assumed to be 50Hz. |
| volatile float | half_period_ms | Half of the period is AC Main's signal in milliseconds. For 60Hz, this value is 8.333ms. For 50Hz, this value is 10ms . |
| float | timer_ticks_per_half_period | The value that the compare register of the timer follows with the AC input wave should be set to match the half period. |
| irq_err_t | zvc_interrupt_open | Indicates whether the ZVC interrupt opens successfully. |
| irq_err_t | hall_interrupt_open | Indicates whether the Hall effect tachometer interrupt opens successfully. |
| bool | isr_init_failed | Indicates whether either of the GPIO interrupts failed to initialize. |

#### 3.3.2.2 triac_t

This data type is meant to store information about an individual TRIAC. This data type is defined in Triac.h

**Table 4. Structure Description and Definition Pertaining to the State of each of the TRIACs**

| Field Type | Name | Description |
|---|---|---|
| triac_identifier_t | id | Identifies the TRIAC by name. As this type is also used to index the global TRIAC array, it is expected that each TRIAC has a unique identifier and that it is added to the global array in the order specified in the enumeration. |
| volatile float | firing_percent | The power percentage of the motor. At 100%, the TRIAC fires immediately after the ZVC. At 0%, it does not power at all. |
| volatile float | rpm | The rotations per minute of the actuator controlled by the TRIAC. This is zero if no measurement data is available. (Measurements taken but not currently linked) |
| volatile float | voltage | The voltage of the actuator controlled by the TRIAC. This is zero if no data is available. (Measurements are taken but not currently linked.) |
| volatile float | current | The current of the actuator controlled by the TRIAC. This is zero if no data is available. (Measurements are taken but not currently linked.) |

### 3.3.3 Global Variables

There are two global variables in this software, which are declared and defined in global_variables.c. They are imported into other files using the global_variables.h header file.

**Table 5. Global Variables**

| Type | Name | Description |
|---|---|---|
| triac_t [3] | TRIACs | Array containing a triac_t for each TRIAC. As indicated previously, the TRIACs are indexed according to their unique id. For example, to edit a field for the medium current TRIAC, the software reads “triacs[MED].xxx”. This relies on the medium current TRIAC being the $0^{th}$ array element. Future software additions must adhere to this convention for the program to function as intended. |
| system_t | system | This is the global instantiation of the of system_t structure. It contains relevant information about the state of the program and its field are mutated as the program advances through its lifecycle. |

### 3.3.4 Global Macros

Located in global_macros.h are constants relating to timer frequency and division, ADCs, reset values, and others. Any change to a clock division requires a corresponding change in this document, as these values are used for timing-based calculations.

## 3.4 Modules

### 3.4.1 TriacControlV2

This TriacControlV2 module acts as the central operating point of the program. It contains the main method and can be traced to understand program flow. The module begins by initializing the relevant peripherals in the init_board() function. It configures the GPIO ports, sets the pins to their correct function, creates a timer for the relay to settle, and initializes the two GPIO-based interrupts using the init_interrupts() function from the System module. If following this function call, the isr_init_failed field of the global system (see Global Variables) does not indicate that the interrupt initialization failed, while the timers and other peripherals required for system measurements and the TRIACs are initialized.

Also, the analog measurements are initialized and started, whereas the digital measurements are only initialized. This distinction is because the digital measurements are GPIO interrupt based, and they have no requirement to be started. On the other hand, analog measurements are based on the periodic reading of the ADCs, so their measurements must be started. If the field indicates that the interrupts were not successfully started, the program is halted.

Next, while there is no halt condition requested, the program runs the system_state_machine() function. This function is what moves the system between operating modes of the device. On BOOT, the state machine starts the relay settling timer, which takes approximately ten seconds. The state is automatically advanced to WAIT_FOR_RELAY, where the status of the relay delay is continuously polled until the relay_delay_complete flag indicates that the delay has concluded. After it has concluded, the state is advanced to CALCULATE_MAIN_FREQUENCY.

The two_zvcs_detected flag is polled continuously until it returns true. This flag indicates whether two voltage crossings have been detected, which means that the system has a timer reading of the time between the crossings and can deduce AC Main's frequency. The program calculates this frequency, starts the watchdog relay pulse, and advances the system state to DRIVE_MOTORS.

In DRIVE_MOTORS, the zvc_detected flag is continuously polled to see if a new ZVC has been detected. If there has been, the half-wave timer is first reset. This timer is set every cycle to expire slightly before the next ZVC is anticipated. At expiration of this timer, the TRIAC gates are no longer pulsed. This construct is necessary so that the pulsing does not bleed into the next cycle, and so that the pulses are stopped if there is no new ZVC detected. After the half_wave timer is reset, the TRIAC gates are scheduled to be pulsed, and the zvc_detected flag is reset. If a halt is requested anywhere in the code, the watchdog relay pulse is stopped, and the program exits.

### 3.4.2 System

#### 3.4.2.1 start_heartbeat()

The start_heartbeat() function simply starts the timer that pulses the watchdog relay.

#### 3.4.2.2 stop_heartbeat()

The stop_heartbeat() function stops the timer that pulses the watchdog relay.

#### 3.4.2.3 calculate_wall_frequency()

The calculate_wall_frequency() function accepts pointers to a volatile bool and volatile float. These are intended to be references to the is_sixty_hertz and half_period_ms fields of the global system structure. The goal of this function is to populate those fields, indicating whether the frequency of AC Main is 60Hz (otherwise assumed to be 50Hz), and the half period of the signal in milliseconds. It first converts the value the count register of CMT2 (the timer that was used to measure the time between the first and second ZVCs) to units of milliseconds. If this value is greater than or equal to the threshold defined in global_macros.h (see Global Macros), it is assumed that the signal is 50Hz. If it is less than the threshold, the signal is assumed to be 60Hz. Depending on the result of that comparison, the half period is set to 8.333ms or 10ms, which are constants defined in global_macros.h.

#### 3.4.2.4 reset_half_wave_timer()

The reset_half_wave_timer() function is called on every ZVC while the state machine is in the DRIVE_MOTORS state. On the first call, it sets the global system timer_ticks_per_half_period. On every succeeding call, it resets the count register of CMT2 to a time that is slightly less than the predicted half wave period. Next, it restarts the timer.

#### 3.4.2.5 start_relay_settling_timer()

The start_relay_settling_timer() function starts the timer associated with the initial relay firing delay. Timer assignments can be viewed in Timers.

#### 3.4.2.6 init_interrupts()

The init_interrupts() Initializes GPIO interrupts associated with the ZVC detector and hall sensor tachometer. If either of them fails, it reflects the failure in the global *system* field and sets the system state to *HALT*.

### 3.4.3 TRIAC Functions

#### 3.4.3.1 set_triac_duty()

The set_triac_duty() function accepts a triac_identifier_t and float, which indicate the TRIAC to fire and at what power. First, a time delay in milliseconds is calculated. This is how long into the half-period the program should wait before beginning to fire the TRIAC gate. For a high power, the gate should be fired sooner. For a low power, it should be fired later. The appropriate timer registers for the delay are then selected based on the TRIAC being fired. The timer is reset, and the delay time is converted to timer counts and put in the appropriate register. If the delay time is within a certain margin of the end of the cycle, the timer is to expire further in the future than it would reach before being reset again. This avoids firing into the next cycle. At that low of a power, the motor likely would not spin anyway.

#### 3.4.3.2 init_triacs()

The init_triacs() function initializes the timers associated with each enabled TRIAC. Includes one for pulse delay, gate drive, and motion profile respectively. Timer assignments can be viewed in Timers.

#### 3.4.3.3 drive_triacs()

For each of the enabled TRIACs, their respective gate pulse timer is stopped, and the set_triac_duty() function is called with the TRIAC parameters, and their pulse delay timer is started. Timer assignments can be viewed in Timers.

### 3.4.4 ZVC

#### 3.4.4.1 zvc_cb()

The zvc_cb() function serves as the callback for the ZVC detection GPIO interrupt. On the first detection, it starts a timer and indicates that next ZVC will be the second. On the second detection, it stops the timer and edits the global system field (see Global Variables) to indicate that two ZVCs have been detected. On every following detection, it simply sets a flag to indicate that a ZVC has been detected.

### 3.4.5 Measurements

#### 3.4.5.1 init_analog_measurements()

The init_analog_measurements() function creates the timer that triggers ADC scanning and initializes the ADC peripherals.

#### 3.4.5.2 start_analog_measurements()

The start_analog_measurements() function starts the timer that triggers the ADC scanning and starts the ADC peripherals.

#### 3.4.5.3 init_digital_measurements()

The init_digital_measurements() function starts the timer that measures the time between hall effect sensor rising edges.

#### 3.4.5.4 hall_cb()

The hall_cb() function stops the timer that measures the time between hall effect sensor edges, converts the measured time to seconds, calculates the RPM of the motor (assuming 2 rising edges per rotation) and restarts the timer. This is used all the callback function for the hall effect sensor interrupt.

## 3.5 Configuration

### 3.5.1 TRIACs

Located in global_macros.h are #defines that enable and disable the three different TRIACs. To enable/disable a specific TRIAC, comment in/out its respective #define.

### 3.5.2 Timers

Central to the operation of this device is timers. Its core functionality revolves pulsing signals at times derived from the AC input wave. For the sake of simplicity, as each TRIAC is firing at a different duty cycle, each TRIAC respectively has three reserved timers. There are also timers reserved for system level tasks, some of which are repurposed at different points in the program. The following describes these roles with Table 6 and Table 7 mapping each timer to its specific TRIAC.

Delay – The time between the detection of the ZVC and when the TRIAC gate should be pulsed

Gate Pulse – The PWM pulse of the TRIAC gate

Motion Profile – Timer to advance between states of motor motion profile (not yet implemented)

**Table 6. Timer Information for Each TRIAC**

| | Delay | Gate Pulse | Motion Profile |
|---|---|---|---|
| Medium Current TRIAC | TMR0 | GPT0 | CMTW0 |
| High Power TRIAC I | TMR2 | GPT1 | CMTW1 |
| High Power TRIAC II | TMR1 | GPT2 | CMT1 |

**Table 7. Timers and Main Function**

| Timer | Function |
|---|---|
| CMT0 | Watchdog relay start delay |
| CMT2 | Measure frequency of AC Main, sine wave safety timer |
| TMR3 | Watchdog pulse |
| TMR4/TMR5 | Hall effect sensor measurement |
| TMR6 | Incremental encoder pulse measurement |
| MTU0 | ADC Scan Trigger |

# 4. Running the TRIAC Board

The TRIAC Motor Inverter Board is designed to plug into the motor control board, as shown in Figure 5 with the relevant hardware interfaces.

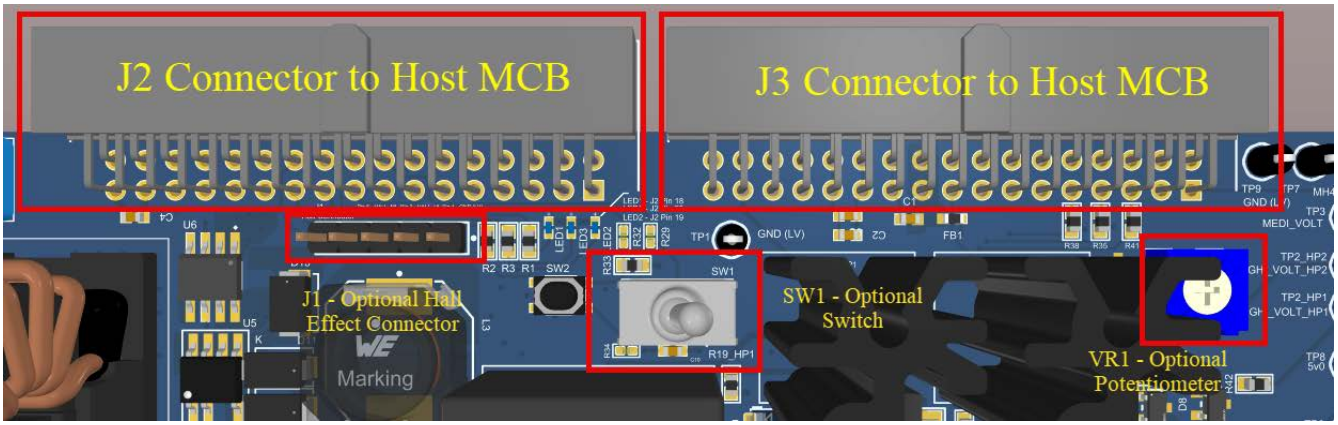



**Figure 5. Board Diagram Outlining Relevant Connectors**

The connectors J2/J3 bridge the hardware signals such as the zero-cross detector, the voltage and current information, watchdog pulse, and the TRIAC driving signals. There are optional hall effect connectors (J1) for extending the functionality of the board, and a switch (SW1, SW2) and potentiometer (VR1).

When importing the project into E2 studio, compiling and debugging the firmware, as outlined in the previous section, the system waits a set amount of time to confirm zero cross detection lock. When achieving frequency lock, an audible click can be heard from the watchdog relay, while the motors can be controlled using the visual expressions command window.

## 4.1 Connectors and Interfacing

A pinout of J1, J2, and the hall effect connector is shown in Table 8.

**Table 8. Invert/Motor Control Board Interface Definition**

| Connector | Pin | Signal Name | Functional Description | Connector | Pin | Signal Name | Functional Description |
|---|---|---|---|---|---|---|---|
| J2 MCB Connector | 1 | MEDI MCU | Firing Signal from the MCU to the Medium Power TRIAC | J3 MCB Connector | 1 | TRIAC_VOLT | Measurement point for the TRIAC Voltage Measurement point, coming from the output of the watchdog relay, used to determine AC Mains power status. |
| | 2 | GND | Ground | | 2 | GND | Ground |
| | 3 | NC | No Connect | | 3 | 5v0+ | 5V rail from the inverter board to the MCU |
| | 4 | GND | Ground | | 4 | GND | Ground |
| | 5 | NC | No Connect | | 5 | IH_ADC | Current Measurement from high power TRIAC 1 |
| | 6 | ZVC_DETECT | Used by the MCU to determine when the sine wave has crossed zero, to time the firing pulse for each half cycle | | 6 | GND | Ground |
| | 7 | NC | No Connect | | 7 | IH2_ADC | Current Measurement from high power TRIAC 2 |
| | 8 | NC | No Connect | | 8 | GND | Ground |
| | 9 | BUS_POWER_IN | Used to detect that the Controller board is connected, enables the 5v regulator for the Inverter board | | 9 | IM_ADC | Current measurement from medium Power TRIAC |
| | 10 | GND | Ground | | 10 | GND | Ground |
| | 11 | GND | Ground | | 11 | HIGH_V | Voltage Measurement from high power TRIAC 1 |
| | 12 | NC | No Connect | | 12 | HIGH2_V | Voltage Measurement from high power TRIAC 2 |
| | 13 | NC | No Connect | | 13 | MED_V | Voltage Measurement from medium power TRIAC |
| | 14 | NC | No Connect | | 14 | GND | Ground |
| | 15 | DRIVER | Periodic Pulse that triggers the watchdog relay on the inverter board | | 15 | VOLT_MEAS | AC Main voltage measurement, used for timing the Zero Voltage Crossing |
| | 16 | ToggleSW | Signal from the Toggle Switch to the MCU | | 16 | NC | No Connect |

| Connector | Pin | Signal Name | Functional Description | Connector | Pin | Signal Name | Functional Description |
|---|---|---|---|---|---|---|---|
| | 17 | PushSW | Signal from the PushButton Switch to the MCU | | 17 | POT | Potentiometer Measurement |
| | 18 | LED1 | LED1 Driver | | 18 | GND | Ground |
| | 19 | LED2 | LED2 Driver | | 19 | 5v0+ | 5V rail from the inverter board to the MCU |
| | 20 | LED3 | LED3 Driver | | 20 | 5v0+ | 5V rail from the inverter board to the MCU |
| | 21 | HALL_HI | Hall effect connector (found on J1) for high power TRIAC Drive 1 | | 21 | GND | Ground |
| | 22 | HALL_HI2 | Hall effect connector (found on J1) for high power TRIAC Drive 2 | | 22 | GND | Ground |
| | 23 | HALL_MI | Hall effect connector (found on J1) for medium power TRIAC Driver | | 23 | 5v0+ | 5V rail from the inverter board to the MCU |
| | 24 | NC | No Connect | | 24 | 5v0+ | 5V rail from the inverter board to the MCU |
| | 25 | NC | No Connect | | 25 | GND | Ground |
| | 26 | NC | No Connect | | 26 | GND | Ground |
| | 27 | NC | No Connect | | 27 | HIGHI_MCU | Firing signal to the high-power TRIAC 1 |
| | 28 | NC | No Connect | | 28 | GND | Ground |
| | 29 | NC | No Connect | | 29 | NC | No Connect |
| | 30 | NC | No Connect | | 30 | GND | Ground |
| | 31 | GND | Ground | | 31 | HIGHI2_MCU | Firing signal to the high-power TRIAC 2 |
| | 32 | GND | Ground | | 32 | GND | Ground |
| | 33 | 5v0+ | 5V rail from the inverter board to the MCU | | 33 | NC | No Connect |
| | 34 | 5v0+ | 5V rail from the inverter board to the MCU | | 34 | GND | Ground |

**Table 9. Hall Connector Table**

| Connector | Pin | Signal Name | Functional Description |
|---|---|---|---|
| **J1 Hall Connector** | 1 | GND | Ground |
| | 2 | 5v0 | 5V rail from the inverter board |
| | 3 | HALL_HI | Hall connector associated with high power TRIAC 1 |
| | 4 | HALL_HI2 | Hall connector associated with high power TRIAC 2 |
| | 5 | HALL_MI | Hall connector associated with medium power TRIAC |

## 4.2 Measurement Points

Various signals are available for measurement from the TRIAC inverter board (see Table 10). Care must be taken to properly isolate the design from the oscilloscope, however, the design is considered hot.

**Table 10. Test Points**

| TP Number | Net Name | Function |
|---|---|---|
| TP1 | GND | Ground Reference |
| TP2_HP2 | HIGHI_VOLT_HP2 | Measurement point for the high-power TRIAC (HP2) |
| TP2_HP1 | HIGHI_VOLT_HP1 | Measurement point for the high-power TRIAC (HP1) |
| TP3 | MEDI_VOLT | Measurement point for the medium power TRIAC |
| TP4 | TRIAC_VOLT_MEASURE | Measurement point for the voltage as controlled by the watchdog relay (divided down). |
| TP5 | VOLT_MEASURE | Measurement point for the voltage being used for Zero Cross Detection (divided down). |
| TP6 | 12v0+ | 12v0 Reference |
| TP7 | GND | Ground Reference |
| TP8 | 5v0+ | 5v0+ Reference |
| TP9 | GND | Ground Reference |

## 4.3 Software Signals and Control

The relevant software signals required to run the TRIAC inverter are described in the following global variables.

- Found in the global system control structure, the halt requested is the emergency stop in the system and immediately stops the firing pulses and the watchdog.

```
    while (HALT != system.state) // Perform normal operation while no HALT has been requested
    {
        system_state_machine ();
    }
```

**Figure 6. Code Snippet Showing the System State Control Variable**

- Found in the triac[] control structure, the member variable firing_percent controls how far into the half sine wave cycle that the firing pulse will fire after a zero cross is detected.

```
/** Stores relevant information about a particular triac **/
typedef struct {
    triac_identifier_t id;
    volatile float firing_percent;
    volatile float rpm;
    volatile float voltage;
    volatile float current;
    volatile float setpoint;
    volatile float kp;
    volatile float ki;
    volatile float kd;
    volatile float last_error;
} triac_t;
    }

//Global arrays storing the information about the TRIACs
triac_t triacs[3] =
{
 {.id = MED, .firing_percent = 0.0, .rpm = 0.0, .voltage = 0.0, .current = 0.0,
.setpoint = 0.0, .kp = 0.0, .ki = 0.0, .kd = 0.0, .last_error = 0.0},

 {.id = HP1, .firing_percent = 0.0, .rpm = 0.0, .voltage = 0.0, .current = 0.0,
.setpoint = 0.0, .kp = 0.0, .ki = 0.0, .kd = 0.0, .last_error = 0.0},

 {.id = HP2, .firing_percent = 0.0, .rpm = 0.1, .voltage = 0.0, .current = 0.0,
.setpoint = 0.0, .kp = 0.00006, .ki = 0.0, .kd = 0.008024, .last_error = 0.0}
};
```

**Figure 7. Code Snippet Showing the TRIAC Control Structures**

These two variables make up the core of the control for the three TRIAC motor drives and can be placed using visual expressions while in e$^2$ studio.

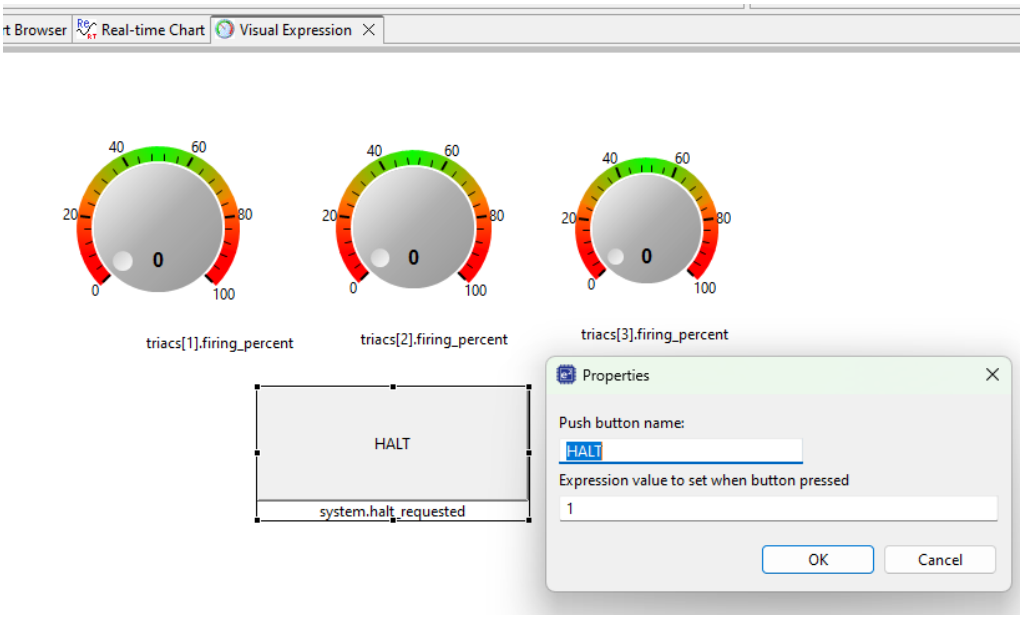



**Figure 8. Example of Visual Expressions Used to Control the TRIAC State Variables**

Using these variables, the user can individually control the firing angle of the three TRIACs resulting in independent control of each motor. A halt command releases the relay causing the motors being decoupled from AC power.

# 5. Revision History

| Revision | Date | Description |
|---|---|---|
| 1.00 | Mar 25, 2026 | Initial release. |

# General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)
   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on
   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state
   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins
   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals
   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin
   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between $V_{IL}$ (Max.) and $V_{IH}$ (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between $V_{IL}$ (Max.) and $V_{IH}$ (Min.).
7. Prohibition of access to reserved addresses
   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products
   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

## IMPORTANT NOTICE AND DISCLAIMER

RENESAS ELECTRONICS CORPORATION AND ITS SUBSIDIARIES (“RENESAS”) PROVIDES TECHNICAL SPECIFICATIONS AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD-PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for developers who are designing with Renesas products. You are solely responsible for (1) selecting the appropriate products for your application, (2) designing, validating, and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. Renesas grants you permission to use these resources only to develop an application that uses Renesas products. Other reproduction or use of these resources is strictly prohibited. No license is granted to any other Renesas intellectual property or to any third-party intellectual property. Renesas disclaims responsibility for, and you will fully indemnify Renesas and its representatives against, any claims, damages, costs, losses, or liabilities arising from your use of these resources. Renesas' products are provided only subject to Renesas' Terms and Conditions of Sale or other applicable terms agreed to in writing. No use of any Renesas resources expands or otherwise alters any applicable warranties or warranty disclaimers for these products.

(Disclaimer Rev.1.01)


### Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com


### Contact Information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit www.renesas.com/contact-us/.

### Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.