TRIAC Control Using iMOTION™ Script Language

Applicable to Heater Control for Hair Dryer

About this document

Scope and purpose

This application note provides examples of how to use the iMOTION™ TRIAC control function for a hair dryer heater element and describes the methods available to the TRIAC control function. The TRIAC control function is only available for FW 5.2 or later; users who want to use this function need to install iMOTION™ Solution Designer (iSD) and program FW5.2 or later to the target device.

Intended audience

This application note is intended for customers who want to understand how to use the iMOTION™ TRIAC control function for their application.

Table of contents

1 Introduction

TRIAC (Triode for Alternating Current) is a type of semiconductor switching device where current can flow in both directions, and widely used for heater control or light dimming applications. One of the target applications of iMOTION™ products is a hair dryer which requires heater control. Integrating the TRIAC control function into iMOTION™ devices is beneficial for system designers of hair dryer application to reduce total BOM and development cost by eliminating external TRIAC control function.

The Motion Control Engine (MCE) firmware supports TRIAC based voltage control through a script plug-in.

1.1 Principle of TRIAC Control

The MCE based TRIAC control system is shown in Figure 1. An AC voltage, V_{AC}, is feeding a load through a TRIAC. The TRIAC's gate pulse is supplied by a gate drive circuit which is driven by a gate pulse from the MCE. To position the gate pulse correctly with respect to the AC-line, the MCE requires information about the AC polarity. That signal is generated by a zero-cross detection circuit. It is necessary to use isolation devices for the zero-crossing detection and the TRIAC gate driver to isolate the high voltage AC-line and the low voltage iMOTION™ controller.

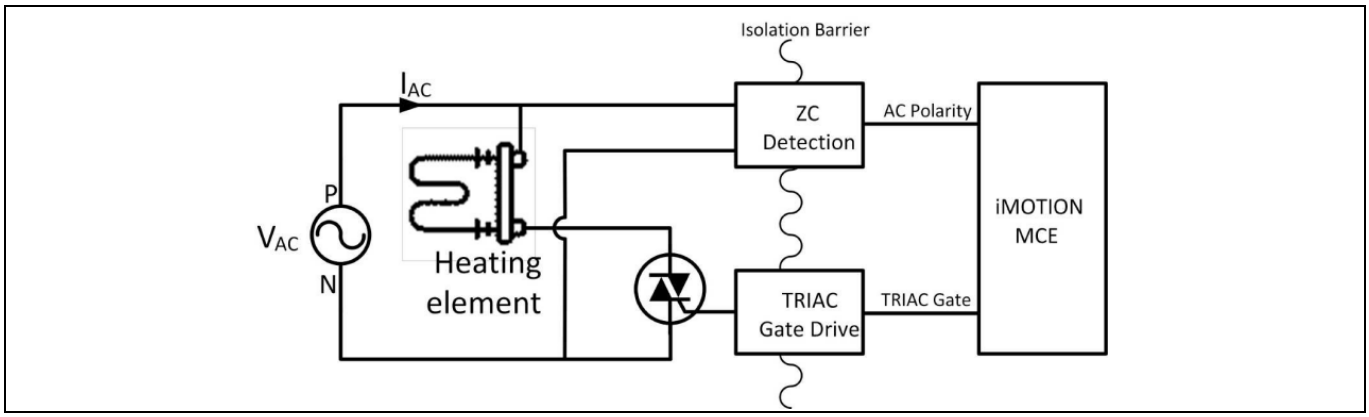


Figure 1 TRIAC Control of a Heating Element

A timing diagram of the TRIAC control is shown in Figure 2. A zero-cross detection circuit monitors the AC-line, V_{AC}, and generates the signal AClineState. Low-high and high-low transitions of the AClineState signal indicates zero-cross moments of the AC-line.

The MCE generates a TRIAC gate pulse which is delayed offTime with respect to the zero-cross of the AC-line and with a pulse width of onTime. The gate pulse turns the TRIAC on and current, I_{AC}, starts to flow through the load. The TRIAC turns off by itself when the current has decayed to zero and sequence starts over.

At offTime = 0, the load sees the full AC-line voltage but as offTime is increased, the voltage across the load is reduced.

The required pulse width of the gate pulse is device specific. With too short a time, the TRIAC may fail to turn fully on.

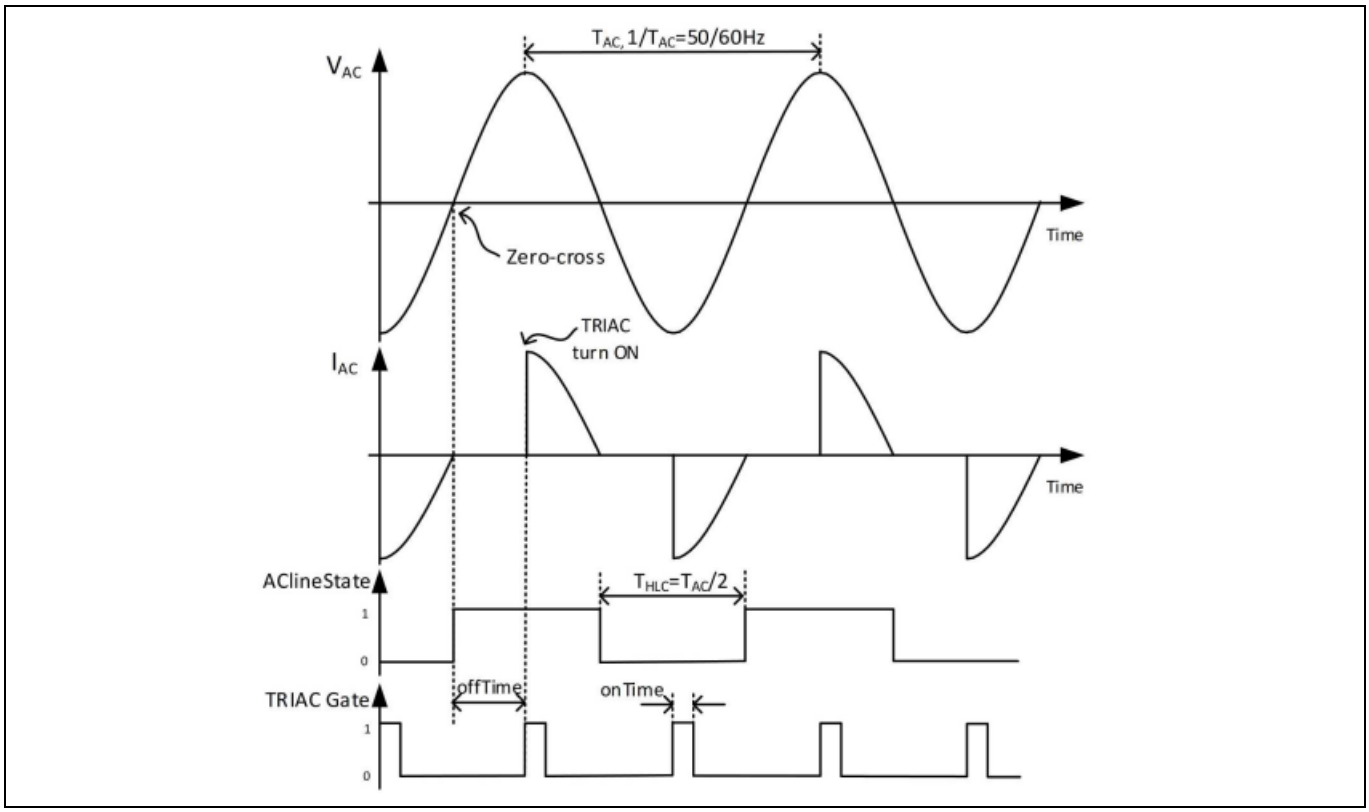


Figure 2 Phase Control of a TRIAC

The user can set the offTime and onTime by API `TRIAC_SetONOFFTime()`, which is described in detail in a later chapter, at any point during a half-line cycle (T_{HLC}). T_{HLC} is a half of AC line cycle time. For example, if the AC input frequency is 50 Hz, $T_{HLC} = \frac{1}{50} \div 2 = 0.01[s]$. The requested offTime and onTime take effect at the following zero-cross of the AC-line as shown in Figure 3.

By latching the onTime and offTime at AC-line zero-cross the update period is kept constant at T_{HLC}. The MCE handles all the time-critical tasks and ensures the correct synchronization to the AC-line. The onTime and offTime can be set with a resolution of 1 µs. The maximum allowed onTime and offTime is 21.84 ms (=21,840 µs).

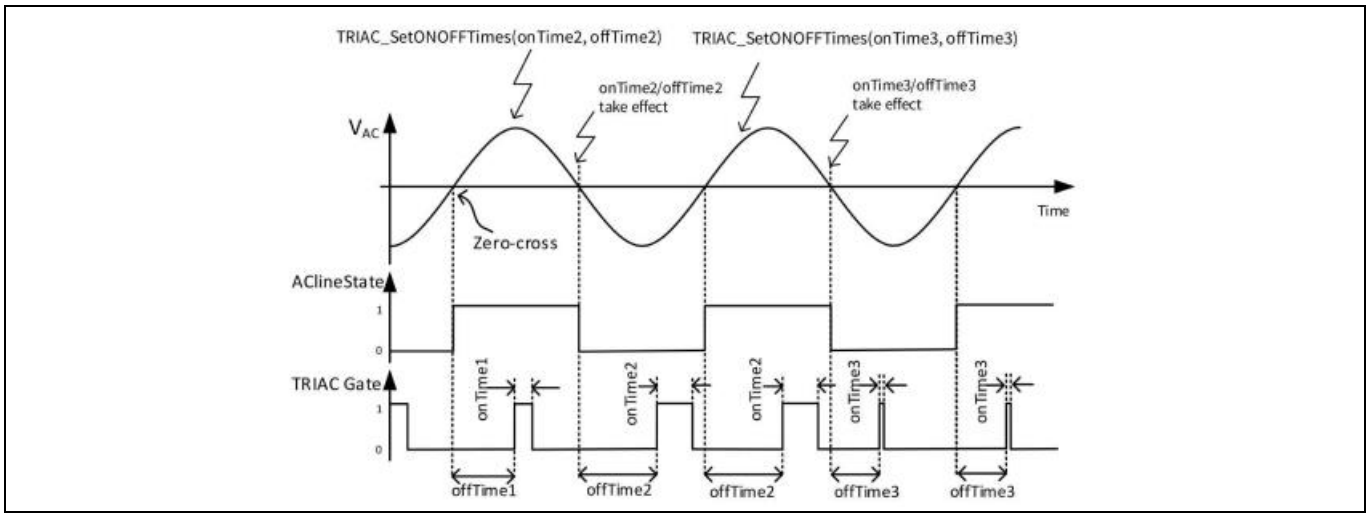


Figure 3 Setting and Latching of Gate Pulse Width and Turn-on Delay

Introduction

With TRIAC control, a new cycle starts at every zero-cross of the AC-line voltage. If the requested offTime and onTime do not fit within T_{HLC}, the gate pulse gets truncated, as shown in Figure 4.

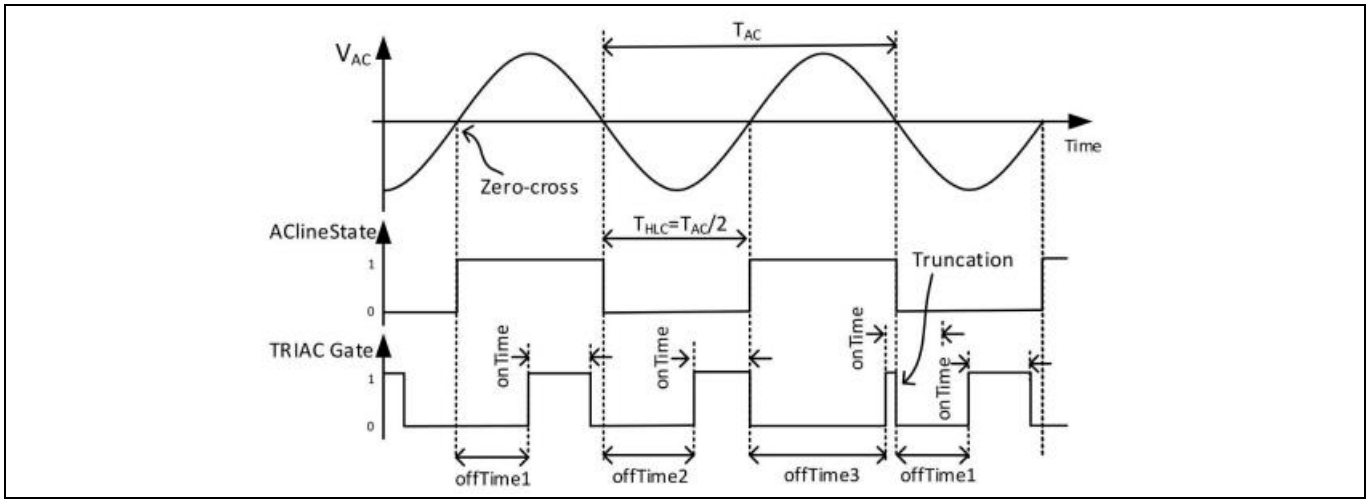


Figure 4 Truncation of Gate Pulse Near Zero-crossing

To avoid truncation, offTime + onTime < T_{HCL}. In Figure 4, offTime1 + onTime1 < T_{HCL} and the gate pulse is completed in time before the next zero-cross. The borderline case is offTime2 + onTime2 = T_{HCL} and no truncation happens in this case. In case of offTime3 + onTime3 > T_{HCL}, the gate pulse is truncated to fit within the half-line cycle.

If offTime > T_{HCL}, no gate pulse is generated, so current does not flow in TRIAC.

1.2 TRIAC Pins

The TRIAC control has one input pin from zero-cross detection circuit, and one output pin to TRIAC gate driver. For flexibility, the MCE offers the option to select between two different input pins (TRIN0 and TRIN1) and two different output pins (TROUT0 and TROUT1). Availability of the TRIAC pins depends on the device and what other functions are supported in the application. Some devices do not support TRIAC control by script because of pin limitation or functionality. Please refer to the datasheet of each iMOTION™ products for pin assignment.

TRIAC input and output pins are selected during initialization of the TRIAC plug-in. The pins are enabled and assigned through the API `TRIAC_DriverInit()`.When using the TRIAC interface, make sure the TRIAC pins are not colliding with other functions and that the pins are not enabled in the User Pin Configuration in iMOTION™ Solution Designer.

The TRIAC input pin has an internal pull-down resistor. The TRIAC output pin is a push-pull stage, Active level of the output pin can be configured by the API `TRIAC_DriverInit()`.

1.3 TRIAC Interface APIs

MCE prepares APIs as a plug in of script function for TRIAC control. The APIs are listed in Table 1. For more information regarding the script engine or TRIAC interface APIs, please refer to [2] and [1], respectively.

Table 1 TRIAC Interface APIs

API name	Brief description
TRIAC_DriverInit()	Initializes the TRIAC interface and initial state of the control
TRIAC_DriverDeInit()	De-initializes TRIAC interface
TRIAC_SetONOFFTimes()	Sets the TRIAC gate on-time and off-time

API name	Brief description
TRIAC_GetStatus()	Returns status bits
TRIAC_ClearStatus()	Clears sticky status bits
TRIAC_GetOnTime()	Returns the actual TRIAC gate on-time
TRIAC_GetOffTime()	Returns the actual TRIAC gate off-time

1.4 System Configuration

Whole system block diagram using iMOTION™ device is shown in Figure 5. AC power is supplied for both main power supply of motor control system and heating element in parallel, so switching behavior of AC power for heating element does not affect main power supply.

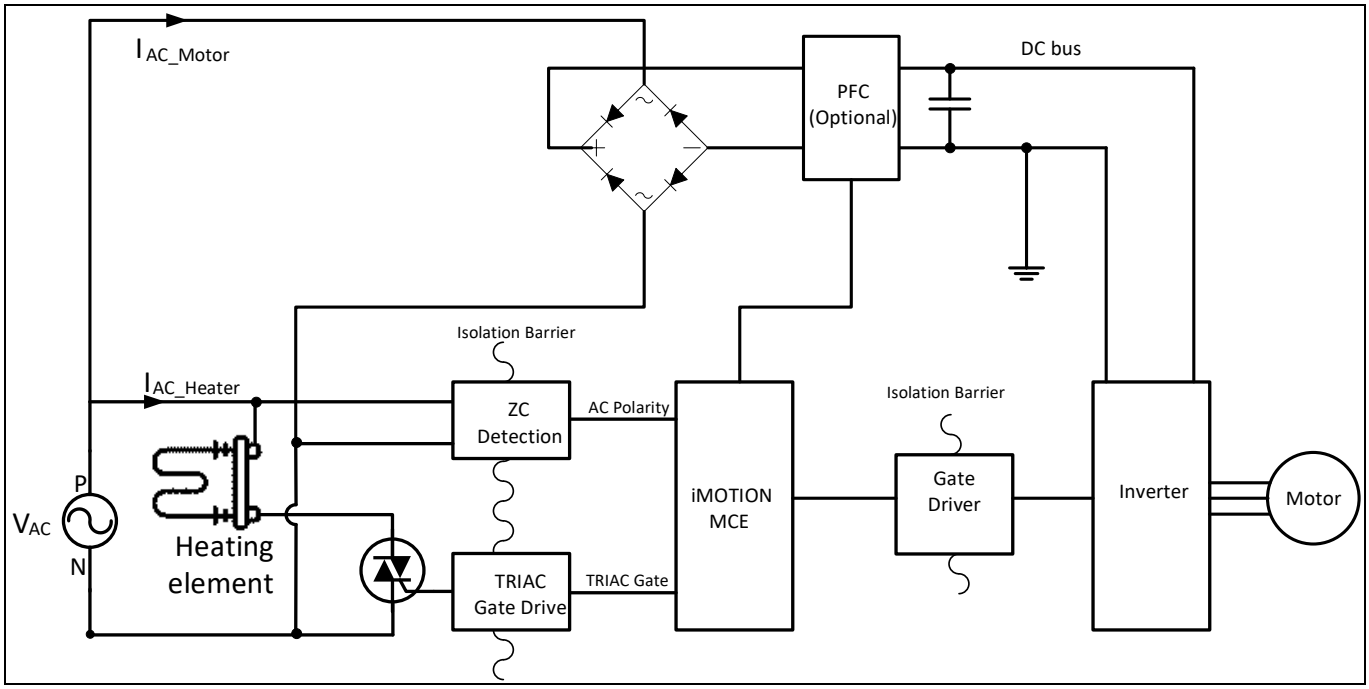


Figure 5 Whole System Block Diagram Including Motor Control and PFC

MCE provides standby modes to enable reduced power consumption of MCE when the motor/PFC are not operating. There are two standby modes, CPU idle sleep and low power mode. TRIAC control function works for both standby modes. Hence it is necessary to control motor/PFC and TRIAC independently. For more information about the standby modes, please refer to [1].

1.5 TRIAC Control Implementation

There are two tasks in MCE script engine; Task0 and Task1. Each task is executed at 1 ms and 10 ms interval, accordingly. It is highly recommended to implement TRIAC control in Task0, because T_{HLC} is usually 10 ms (f_{AC} = 50 Hz) and 8.3 ms (f_{AC} = 60 Hz).

In order to start TRAIC control, `TRIAC_DriverInit()` API needs to be called to initialize TRIAC control driver to specify TRIAC input and output pins, and initial onTime and offTime. If it is necessary to change offTime and onTime after initialization, `TRIAC_SetONOFFTime()` API needs to be called.

2 TRIAC Control Example by Script Engine

2.1 Overview

In this chapter, example code for TRIAC control is described. Figure 6 shows the system block diagram. The zero-cross detection circuit and the TRIAC gate driver are connected to the iMOTION™ device. In addition, an external DC power supply is connected to AIN0 pin to set the desired off-time. It means that current through the heating element, as well as temperature of heating element, is controlled by the voltage applied to AIN0 pin. The external DC power supply can be realized by resistive voltage divider circuit.

Note: To filter noise on the TRIN pin and avoid unexpected switching of the TRIAC, it is recommended to add a small bypass capacitor to the input line from Zero-Cross Detection Circuit (TRIN) at a location close to the iMOTION™ device.

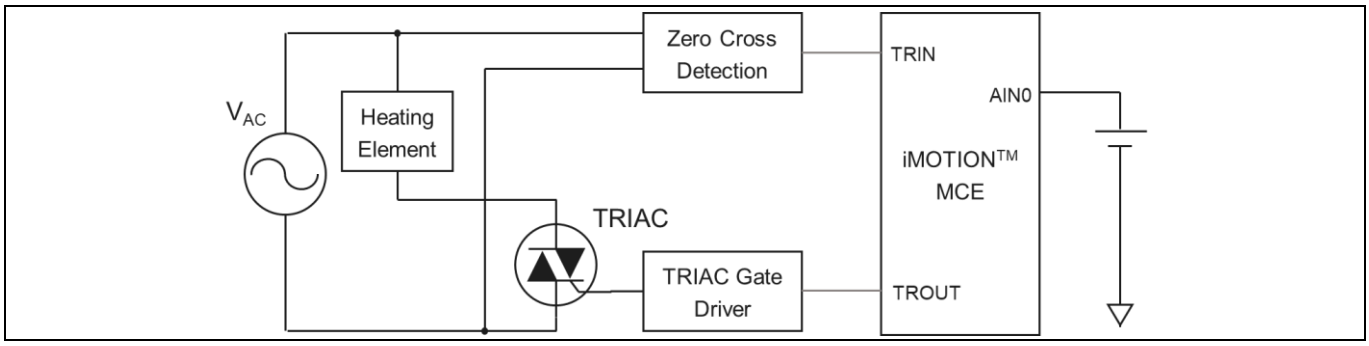


Figure 6 System Block Diagram of Example

To make the AIN0 pin accessible from the script, it is necessary to configure **User Pin Configuration** in the Configuration Wizard in iSD. Please check AIN0 pin as shown in Figure 7.

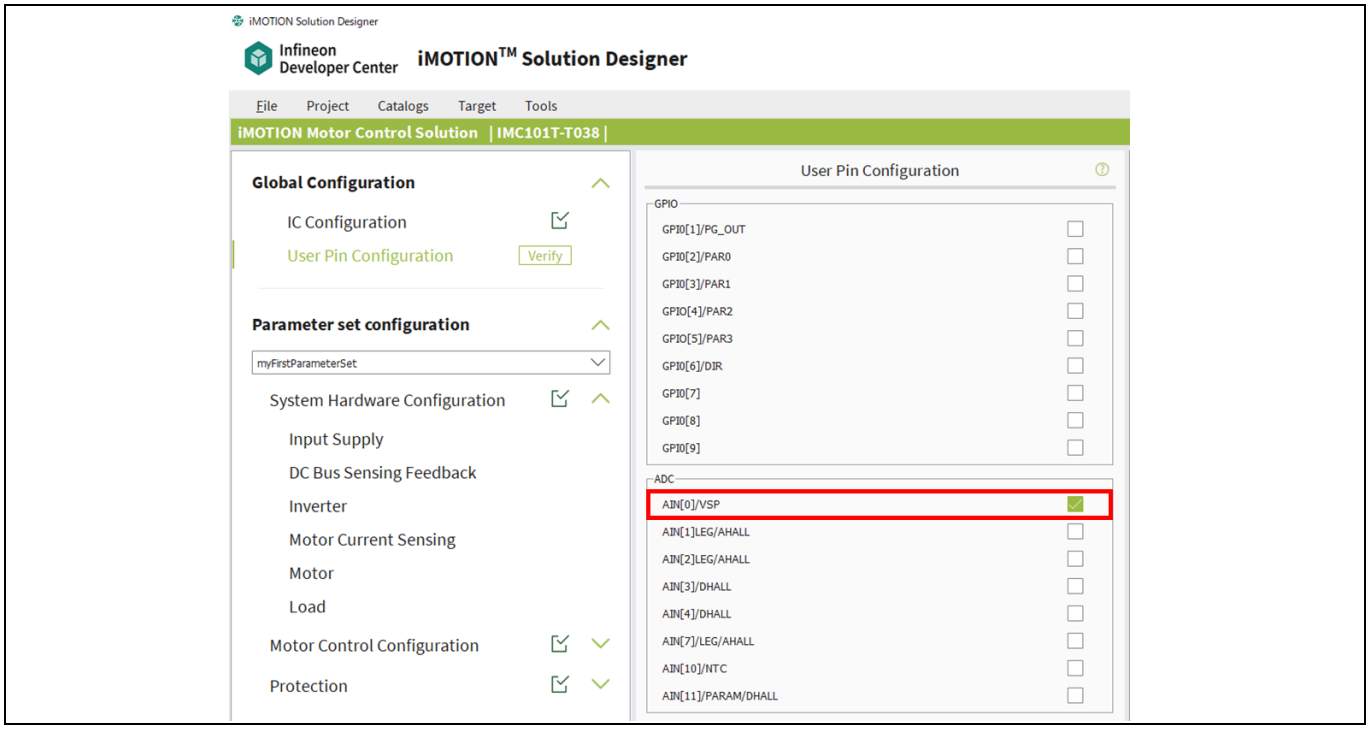


Figure 7 User Pin Configuration

TRIAC Control Example by Script Engine

All functions are implemented in script Task0 which is executed every 1 ms. Please set SCRIPT_TASK0_EXECUTION_PERIOD to 1 in **Properties** window in the Script Editor in iSD as shown in Figure 8, so that Task0 runs at 1 ms interval.

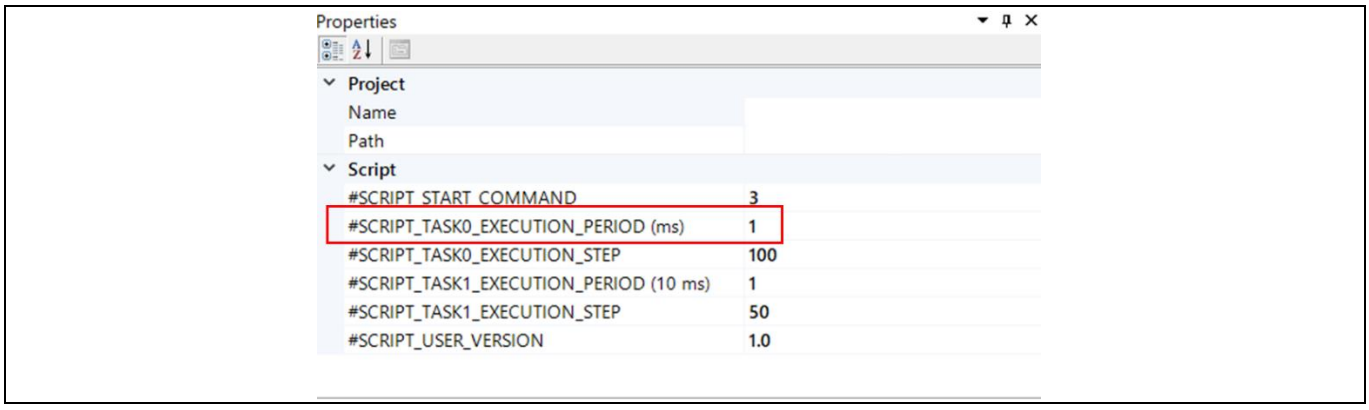


Figure 8 Setting of Properties Window in Script Editor

Figure 9 shows the control flow chart of the TRAIC control by script engine.

There are two state machines; control_state and heater_state. control_state defines which control function is executed (detecting and calculating THLC or TRIAC control). heater_state controls the heater temperature by changing the offset time of gate pulse. heater_state is controlled by external voltage applied to the AIN0 pin.

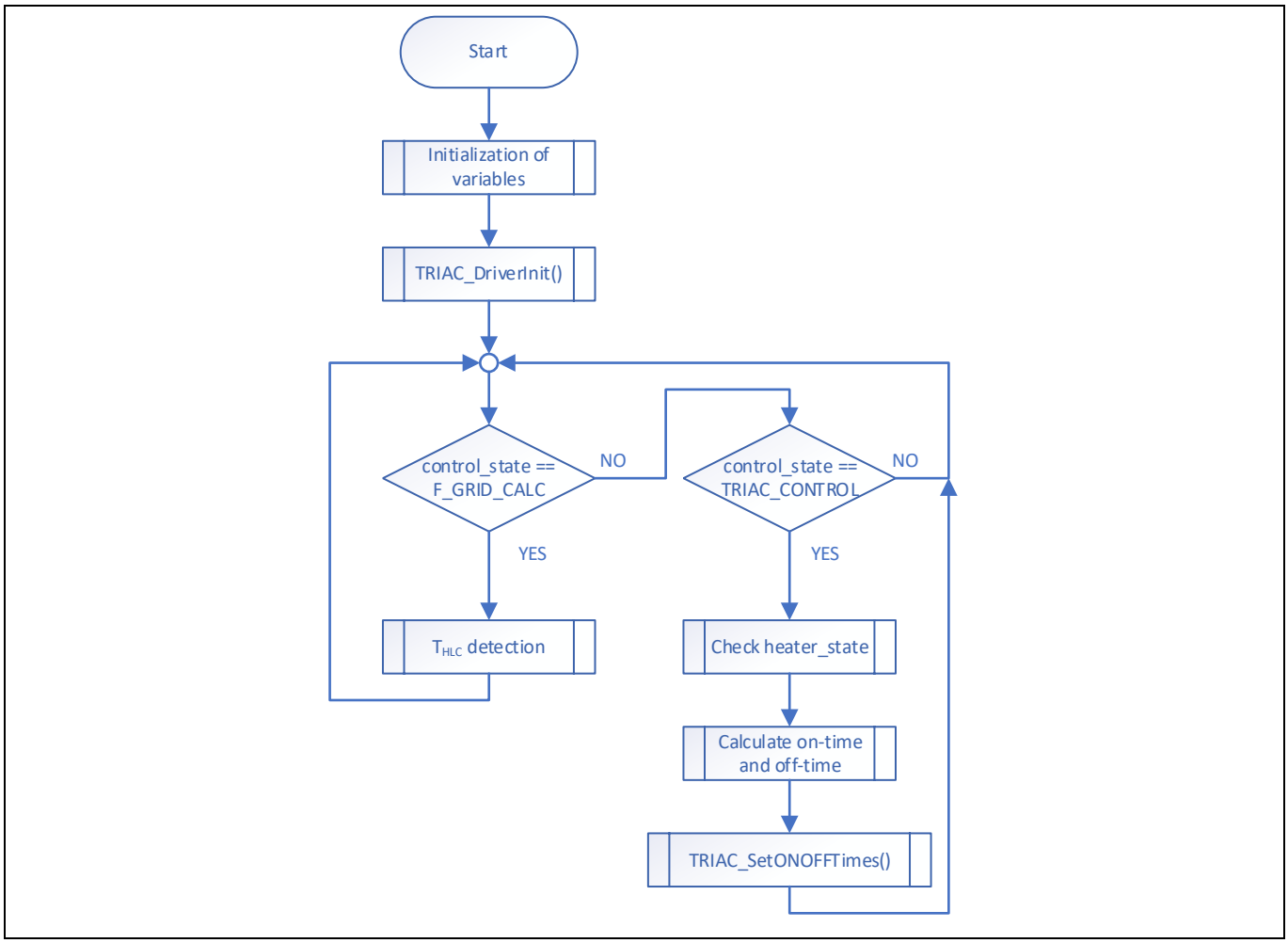


Figure 9 TRIAC Control Flow Chart

2.2 State Machines

2.2.1 Control State

Control state defines which control function is executed; T_{HLC} Calculation or TRIAC control. After initialization, control state is set to T_{HLC} calculation. After T_{HLC} calculation is done, control state is changed to TRIAC control. The offset adjustment of the gate pulse is done in TRIAC control state according to the calculated T_{HLC}.

2.2.1.1 T_{HLC} Calculation

The AC-line frequency of commercial power supply is different by regions in some country like Japan. In such case, the TRAIC off-time needs to be adjusted according to the AC-line frequency.

In this example, T_{HLC} is measured after the script code has been initialized but before the actual TRIAC control has started. The interval between zero-cross events is measured `NUM_OF_ZC_DETEC` times, after which T_{HLC} is calculated as an average of the interval. For the TRIAC control the off-time is adjusted based on the calculated T_{HLC} value.

Figure 10 shows the flowchart to calculate T_{HLC}. As a default, parameter timer flag is set at TIMER_STOP. When Task0 is executed for the first time, RunTimeCounter value is set to start_time, and timer_flag is changed to TIMER_RUN. Task0 checks the zero-cross status by using the TRIAC_GetStatus API. If a zero-cross is detected, a sticky bit of TRIAC_GetStatus() return value is cleared by calling the TRIAC_ClearStatus API and counter register ZC_cnt is incremented. When ZC_cnt reaches NUM_OF_ZC_DETEC, RunTimeCounter value is set to register delta_time with delta_time calculated as end_time – start_time, in the unit of [ms]. Finally. T_{HLC} (register name half_line_cycle) is calculated by the following equation. Unit of half_line_cycles is [µs].

$$half\_line\_cycle = \frac{(delta\_time * 1000)}{NUM\_OF\_DETEC}$$

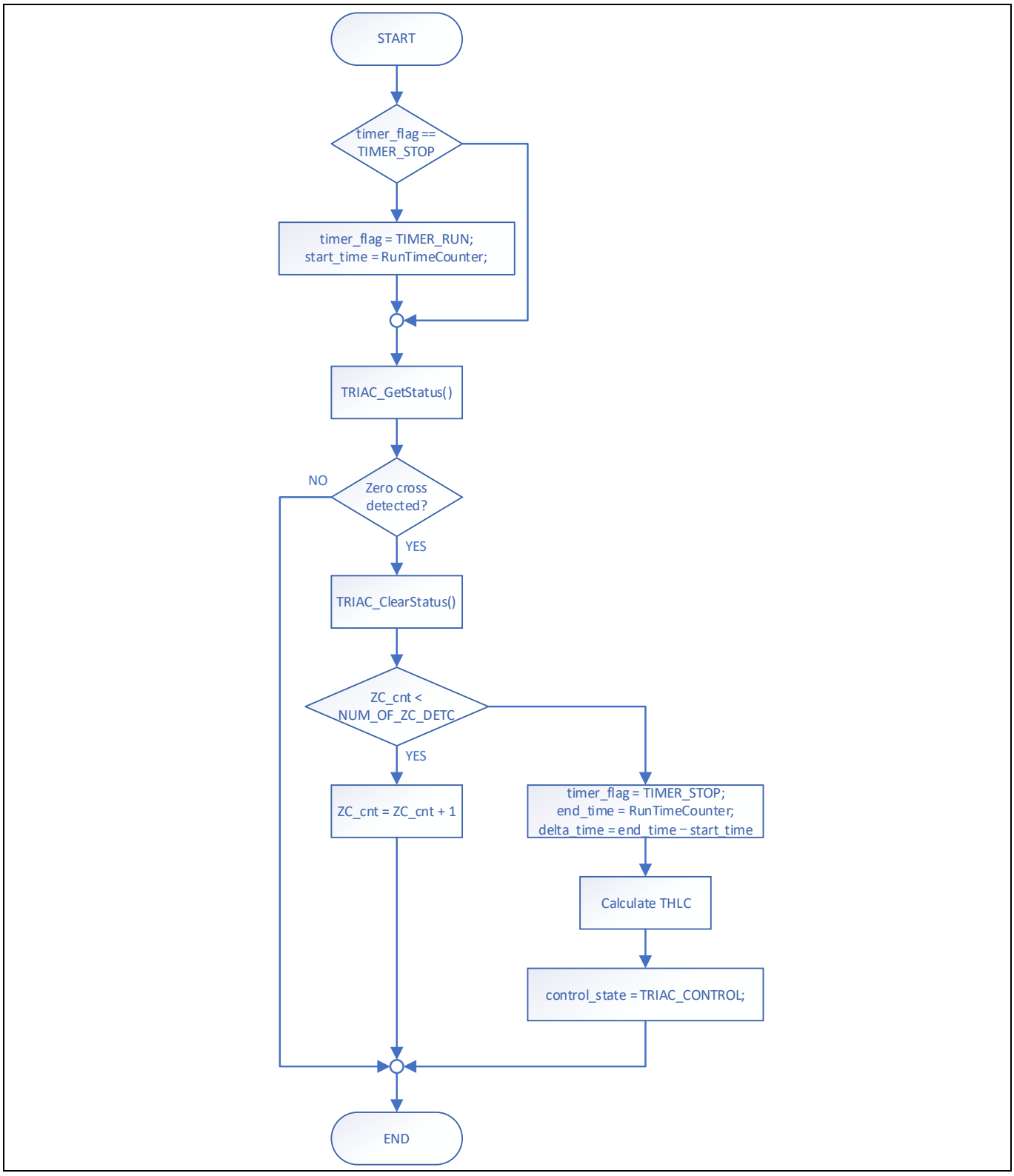


Figure 10 Flowchart to Calculate T_{HLC}

After T_{HLC} calculation is finished, control_state register is set to TRAIC_CONTROL, and T_{HLC} detection process will not be executed again.

If it is not necessary to adapt to different AC-line frequencies, T_{HLC} and off-time can be set to predetermined fixed values.

2.2.1.2 TRIAC Control and Offset Adjustment

When control_state is TRIAC_CONTROL, offTime shown in Figure 2 is changed to control heater temperature. The offTime is set according to an external voltage applied to AIN0 pin, as well as T_{HLC}.

In this example, heater_duty is used for duty ratio of AC output in the unit of [%]. For example, if duty ratio is 30%, TRIAC is ON for 30% of total duration of the AC half-line cycle. In this case, offTime is 70% (=100 – 30) of half-line cycle. The offTime should be adjusted according to the actual half-line cycle and calculated by the following equation in the unit of [µs]. The unit of half_line_cycle is also [µs] as described in 2.2.1.1.

$$offTime = half\_line\_cycle * (100 - heater\_duty) / 100$$

2.2.2 Heater State

Heater state defines off-time of the gate pulse to the TRIAC gate driver. Figure 11 shows the relationship between the voltage at the AIN0 pin and heater state. Smaller offTime leads higher current flow in the heater element. In this example, heater state has 3 active levels (HIGH, MID, LOW) and STOP level. The state is changed based on voltage applied to the AIN0 pin. Hysteresis is added to avoid chattering between the states. In the example code, the duty ratio for each state is 0%, 30%, 60%, 100% for STOP, LOW, MiD and HIGH states, respectively.

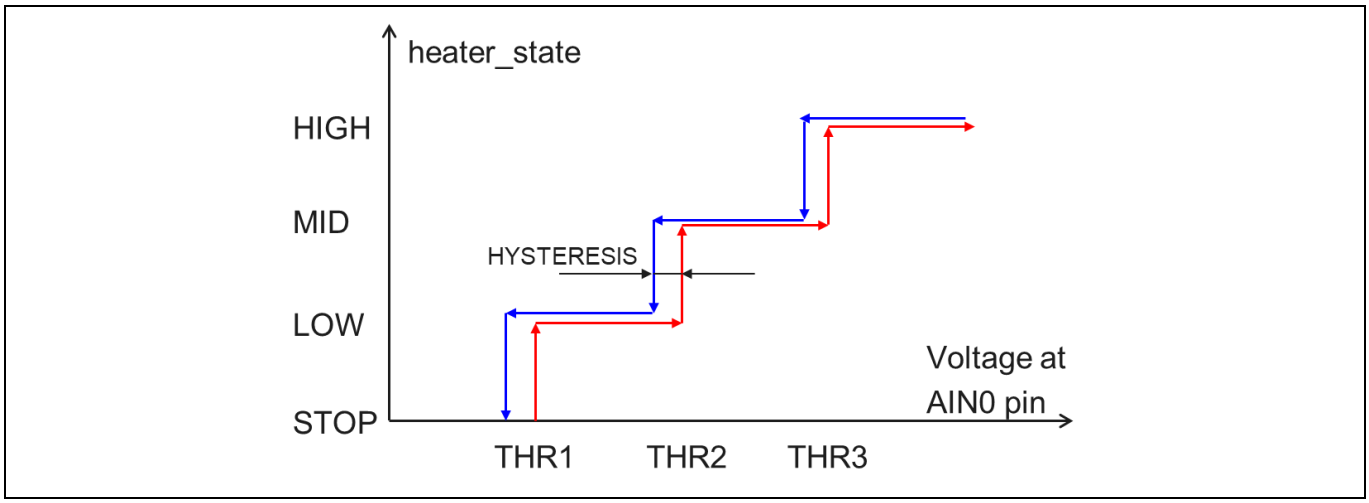


Figure 11 Relationship between Voltage at AIN0 Pin and Heater State

onTime and offTime changes according to heater_state. When heater_state is LOW or MID, offTime is calculated as shown in 2.2.1.2. When the duty ratio is zero, gate pulse should not be generated. In this case, it is necessary to configure onTime = 0 and offTime = 0. When duty ratio is 100%, it is necessary to keep the gate constantly ON. In this case, it is necessary to configure onTime = 0xFFFF and offTime = 0.

After onTime and offTime adjustments are done, `TRIAC_SetONOFFTime()` API is called to set new the onTime and offTime. The new onTime and offTime become effective from the next half line cycle after the API is called.

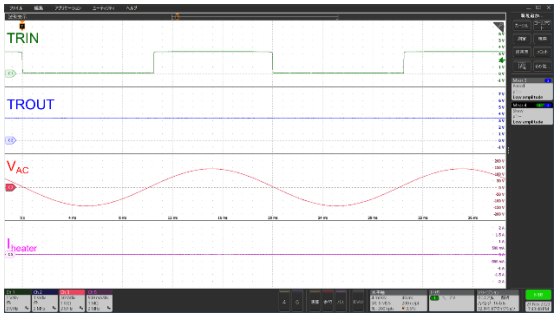
2.3 TRIAC Control Operation Example

Figure 12 shows the example of TRIAC operation, by changing external voltage applied to AIN0 pin. When AC input voltage (V_{AC}) is positive, TRIN signal becomes HIGH. When V_{AC} is negative, TRIN signal becomes LOW.

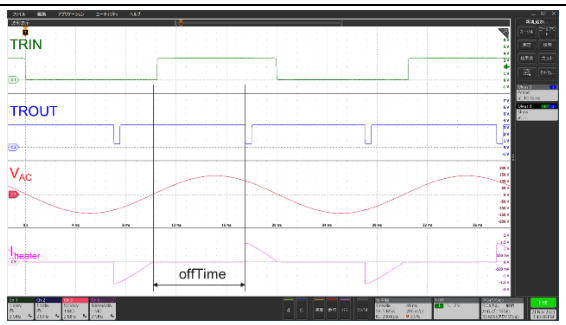
When duty ratio is zero, gate is kept constantly OFF. So heater current (I_{heater}) does not flow as shown in Figure 12 (a).

offTime of gate pulse changes as duty ratio changes, as shown in Figure 12 (b) and (c).

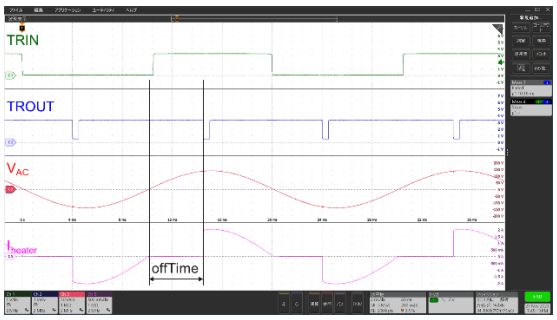
When duty ratio is 100%, gate is kept constantly ON. So I_{heater} flows during whole half line cycle time, as shown in Figure 12 (d).



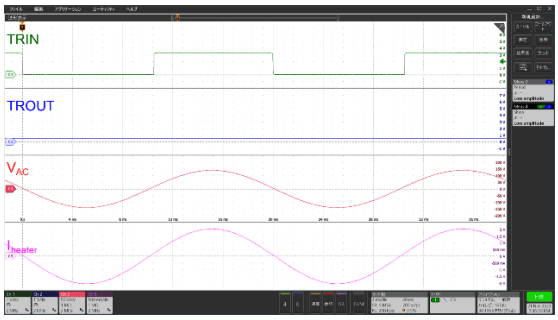
(a) STOP state (duty = 0%)



(b) LOW state (duty = 30%)



(c) MID state (duty = 60%)



(b) HIGH state (duty = 100%)

Figure 12 TRIAC Control Operation Example

2.4 Script Code Example for TRIAC Control

Code listing 1 Definition of constants in Global.mcs

```
const int TRIAC_STATUS_ZERO_CROSSING_MASK   = 0x01; /* Bit 0 - Zero Crossing */
const int THLC_MEASURE = 0;
const int TRIAC_CONTROL = 1;
const int TIMER_STOP = 0;
const int TIMER_RUN  = 1;
const int NUM_OF_ZC_DETEC = 100;
/* TRIAC Driver Initialization */
const int TRIN = 0;
const int TROUT = 0;
const int OUTPUT_INV = 1;
/* heater state */
const int HEATER_STOP = 0;
const int HEATER_LOW = 1;
const int HEATER_MID = 2;
const int HEATER_HIGH = 3;
/* heater duty in % */
const int HEATER_DUTY_STOP = 0;
const int HEATER_DUTY_LOW = 30;
const int HEATER_DUTY_MID = 60;
const int HEATER_DUTY_HIGH = 100;
/* AIN0 thershold and hysteresis */
const int HEATER_THR1 = 992;   /* 0.8 V */
const int HEATER_THR2 = 2233;   /* 1.8 V */
const int HEATER_THR3 = 3474;   /* 2.8 V */
const int HEATER_HYSTERESIS = 248; /* 0.2 V */
/* Default onTime and offTime */
const int GATE_ON_TIME = 500;   /* [us] */
const int DELAY_TIME = 1000;   /* [us] */
```

Code listing 2 Definition of Global Variables in Global.mcs

```
uint16_t on_time, off_time;
uint8_t control_state, heater_state;
uint8_t timer_flag;
uint16_t heater_duty;
uint8_t triacStatus;
uint8_t ZeroCrossFlag;
uint16_t ZC_cnt;
uint16_t half_line_cycle; /* [us] */
int32_t start_time, end_time, delta_time;
uint16_t ain0_val;
```

Code listing 3 Script_Task0_init()

```
001        Script_Task0_init()
002        {
003           uint16_t ClearStatus;
004
005           off_time = 0;
006           on_time = 0;
007           ZC_cnt = 0;
008           half_line_cycle = 0;
009           start_time = 0;
010           delta_time = 0;
011           control_state = THLC_MEASURE;
012           timer_flag = TIMER_STOP;
013           heater_duty = HEATER_DUTY_STOP;
014           on_time = GATE_ON_TIME;
015           off_time = heater_duty;
016           TRIAC_DriverInit(off_time, on_time, TRIN, TROUT,
   OUTPUT_INV);
017        }
```

Code listing 4 Script_Task0()

```
001        Script_Task0()
002        {
003           /* Acquire VSP analog voltage */
004           ain0_val = FB_ADC.adc_result[0];
005           if (control_state == THLC_MEASURE)
006           {
007              if (timer_flag == TIMER_STOP)
008              {
009                 timer_flag = TIMER_RUN;
010                 start_time = MCEOS.RunTimeCounter;
011              }
012              triacStatus = TRIAC_GetStatus();
013                ZeroCrossFlag = triacStatus &
   TRIAC_STATUS_ZERO_CROSSING_MASK;
014              if (ZeroCrossFlag)          /* Clear zero-cross flag */
015              {
016                   ClearStatus = TRIAC_STATUS_ZERO_CROSSING_MASK;
017                   TRIAC_ClearStatus(ClearStatus);
018                   if (ZC_cnt < NUM_OF_ZC_DETEC)
019                   {
020                       ZC_cnt = ZC_cnt + 1;
021                   }
022                   else   // ZC_cnt == (NUM_OF_ZC_DETEC +1))
023                   {
024                      end_time = MCEOS.RunTimeCounter;
025                      delta_time = end_time - start_time;
026                      half_line_cycle = (delta_time *
   1000)/NUM_OF_ZC_DETEC; // half line cycle in us
027                      control_state = TRIAC_CONTROL;
028                      timer_flag = TIMER_STOP;
029                      on_time = GATE_ON_TIME;
030                   }
031              }
032           }
```

```
033             if (control_state == TRIAC_CONTROL)
034             {
035                 /* heater state */
036                 if (heater_state == HEATER_STOP)
037                 {
038                     if (ain0_val > HEATER_THR1)
039                     {
040                         heater_state = HEATER_LOW;
041                         heater_duty = HEATER_DUTY_LOW;
042                     }
043                 }
044                 if (heater_state == HEATER_LOW)
045                 {
046                     if (ain0_val > HEATER_THR2)
047                     {
048                         heater_state = HEATER_MID;
049                         heater_duty = HEATER_DUTY_MID;
050                     }
051                     if (ain0_val < HEATER_THR1 - HEATER_HYSTERESIS)
052                     {
053                         heater_state = HEATER_STOP;
054                         heater_duty = HEATER_DUTY_STOP;
055                     }
056                 }
057                 if (heater_state == HEATER_MID)
058                 {
059                     if (ain0_val > HEATER_THR3)
060                     {
061                         heater_state = HEATER_HIGH;
062                         heater_duty = HEATER_DUTY_HIGH;
063                     }
064                     if (ain0_val < HEATER_THR2 - HEATER_HYSTERESIS)
065                     {
066                         heater_state = HEATER_LOW;
067                         heater_duty = HEATER_DUTY_LOW;
068                     }
069                 }
070                 if (heater_state == HEATER_HIGH)
071                 {
072                     if (ain0_val < HEATER_THR3 - HEATER_HYSTERESIS)
073                     {
074                         heater_state = HEATER_MID;
075                         heater_duty = HEATER_DUTY_MID;
076                     }
077                 }
078                  if (heater_duty == 0)
079                 {
080                     on_time = 0;
081                     off_time = 0;
082                 }
083                 else
084                 {
085                     if (heater_duty >= 100)
086                     {
087                         on_time = 0xFFFF;
088                         off_time = 0;
```

```
089                 }
090                 else
091                 {
092                     on_time = GATE_ON_TIME;
093                     off_time = half_line_cycle * (100 - heater_duty) /
    100;
094                 }
095             }
096             TRIAC_SetONOFFTimes(on_time, off_time);
097         }
098     }
```

3 TRIAC Control Design Guideline

- Please use Script Task0 for implementation of TRIAC control. Execution period of Script Task1 is too slow (10 ms) to control TRIAC.
- To filter noise on the TRIN pin and avoid unexpected switching of the TRIAC, it is recommended to add a small bypass capacitor to the input line from Zero-Cross Detection Circuit (TRIN) at a location close to the iMOTION™ device.
- Please refer to the datasheet of TRIAC device when setting onTime.
- In order to avoid truncation, please make sure offTime + onTime < T_{HCL}.
- To keep the gate constantly OFF, please set onTime = 0 and offTime = 0. To keep the gate constantly ON, please set onTime = 0xFFFF and offTime = 0.

4 References

[1] iMOTION™ Motion Control Engine Functional Reference Manual

[2] AN2018-27: How to Use iMOTION™ Script Language

[3] iMOTION Solution Designer User Guide

[4] Getting Started Guide with iMOTION™ Solution Designer

Revision history

Document version	Date of release	Description of changes
1.0	2023-12-25	Initial release

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Edition 2023-12-25

Published by

Infineon Technologies AG

81726 Munich, Germany

© 2024 Infineon Technologies AG. All Rights Reserved.

Do you have a question about this document?

Email: erratum@infineon.com

Document reference
AN2023-09

IMPORTANT NOTICE

The information contained in this application note is given as a hint for the implementation of the product only and shall in no event be regarded as a description or warranty of a certain functionality, condition or quality of the product. Before implementation of the product, the recipient of this application note must verify any function and other technical information given herein in the real application. Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind (including without limitation warranties of non-infringement of intellectual property rights of any third party) with respect to any and all information given in this application note.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

For further information on the product, technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies office (**www.infineon.com**).

WARNINGS

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.