NXP Semiconductors
Application Note

Document Number: AN5355
Rev. 0, 11/2016

Interleaved PFC Average Current Control

Implementation using MKV46F256VLx16 on High Voltage Motor Control Platform

1. Introduction

This application note describes the implementation of the interleaved Power Factor Correction (PFC) operating in the Continuous Conduction Mode (CCM) on the MKV46F256VLx16 ARM® Cortex®-M4 MCU. The MKV46F256VLx16 is a member of the Kinetis KV4x MCU family. The peripherals available on this MCU are dedicated for an easy implementation of the power conversion applications.

A two-channel interleaved PFC operating in the CCM mode is used for the AC/DC conversion while compensating the Power Factor (PF). This application note includes the system design concept, the software design, and the control loops design. The detailed system operation, the PFC control theory, and the control loops design is described in *Average Current Mode Interleaved PFC Control* (document AN5257). All equations mentioned in this application note refer to AN5257.

The hardware used for the PFC application development is the High Voltage Motor Control Platform (HVP-MC3PH) together with the MKV46F150M Controller Card (HVP-KV46F150M). Both boards are a part of the NXP High Voltage Development Platform. All information about this development platform are available at nxp.com/hvp.

Contents

2. System design concept

This chapter describes the hardware used for the PFC application implementation and the application control process implementation on the MKV46F150M controller.

2.1. System architecture

The application is developed on the High Voltage Motor Control Platform (high-voltage power stage HVP-MC3PH) and the KV46F150M Controller Card (HVP-MKV46F150M). The system design architecture is shown in this figure:

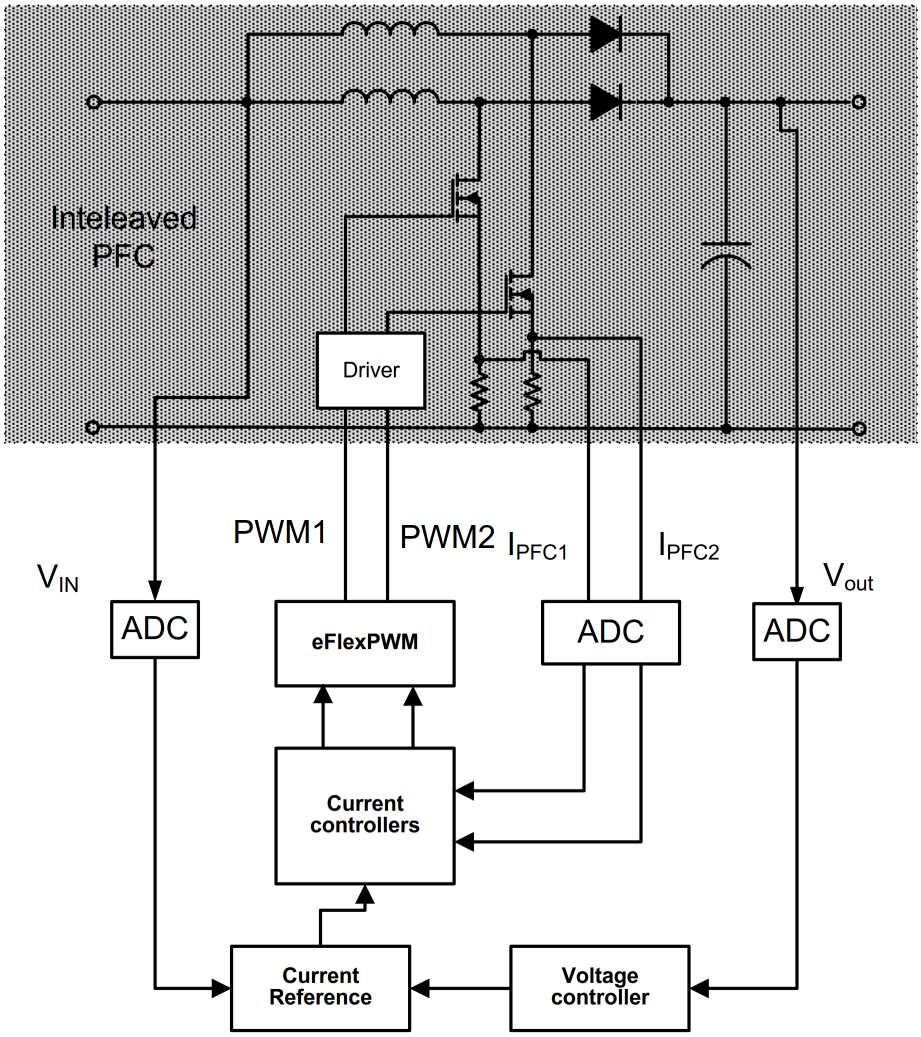

Figure 1. System design architecture

The high-voltage power stage contains the whole power circuitry and the signal conditioning stage for the adaptation of the sensing signals for the controller card. The high-voltage power stage and the controller card are described in detail in the *Freescale High-Voltage Motor Control Platform User's Guide* (document HVPMC3PHUG) and the *HVP-KV46F150M User's Guide* (document HVPKV46F150MUG).

CAUTION

Before you run the application, check the revision of your HVP-MC3PH board. If your board revision is 1, please replace the operation amplifier U2 by MAX4477ASA+. Otherwise, the PFC application is not going to run properly.

2.2. System specification

The application has these performance features:

1. Hardware topology used:
   - Interleaved boost converter.
2. Control technique incorporated:
   - Average current mode control with a constant switching frequency of 100 kHz.
   - 20-µs current loop.
   - 1-ms voltage loop.
   - Soft start.
3. Fault protection:
   - Input under-voltage and over-voltage.
   - DC-Bus under-voltage and over-voltage.
   - Over-current.
   - Start-up fault.

2.3. PFC control process implementation on KV46

The PFC control process consists of the implementation of two control loops, as shown in Figure 2: the fast current control loop and the slow voltage control loop. The output DC-Bus voltage (V_{out}) is compared with the reference voltage ($V_{out\ reference}$). The voltage difference (V_{out_error}) is fed to the voltage controller. The voltage controller output is multiplied by the input voltage (V_{in}) and the feed-forward factor, which is derived from the RMS value of the input voltage. The result of the multiplication is the current reference (I_{ref}) for both current loops. Two boost converters operate independently and therefore two current control loops are implemented. The output current errors (I_{1err} and I_{2err}) are then processed in the current controllers. The outputs from the current controllers are the duty cycles for the boost converters. The interleaved PFC topology enables the operation of two converters in parallel, sharing the converter load by a half.

The key peripherals for the PFC application implementations are the Analog-to-Digital Converter (ADC), the Pulse Width Modulator (eFlexPWM), the Periodic Interrupt Timer (PIT), the Inter-Peripheral Crossbar Switch (XBAR) and the And/Or/Invert logic module. The PWM submodule 3 generates the driving signals for two MOSFET transistors and the PWM is configured to generate center-aligned PWM signals. The PWM reload interrupt is generated every second switching period (20 µs). This fast interrupt processes both current control loops. The generated PWM signals are shifted by a half of the switching period. The ADC senses the analog quantities in the middle of each PWM period and each phase. The sensed quantities are the input voltage, the boost converter currents, and the

output DC-Bus voltage. The measurement is synchronized with the PWM generation due to the measurement of the average values of the boost currents. The PIT timer 0 is used to generate the 1-ms periodical interrupt for the voltage control loop processing. Two trigger signals (one for each phase) are generated by the eFlexPWM module in the middle of each PWM period and linked to the ADC via the XBAR module. The XBAR module serves to interconnect the control signals between the peripherals. The communication with FreeMASTER is done via the UART serial interface module.

FreeMASTER is a run-time debugging/tuning tool for application development and information management. The peripheral configurations are described in detail in Section 3.7, "PFC Control Peripheral Drivers".

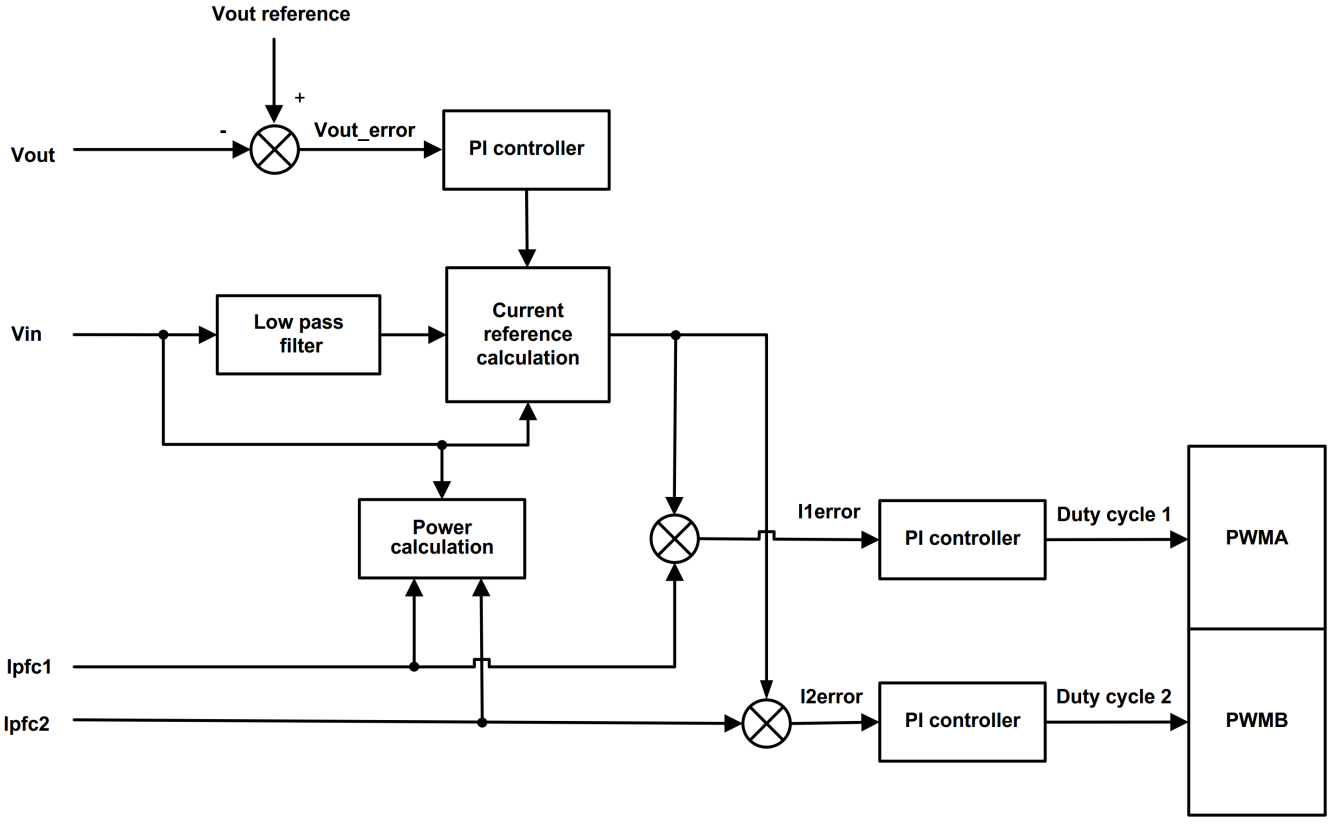

Figure 2. System control scheme

3. Software design

This chapter describes the software design of the interleaved PFC application. The project was developed in the IAR Embedded Workbench® IDE using the Kinetis SDK IDE and the Real-Time Control Embedded Libraries (RTCESL).

3.1. Project files structure

The project is located in three sub-directories and consists of these files:

1. *..\build\iar\hvp_kv46_pfc_ccm_interleaved.eww*—the workspace data file for the IAR Embedded Workbench IDE, double-click the file to launch the IAR IDE.

2. ..*build**iar**hvp_kv46_pfc_ccm_interleaved.ewp*—the project file for the IAR Embedded Workbench IDE.
3. ..*build**iar**hvp_kv46_pfc_ccm_interleaved.ewd*—the debugger data file for the IAR Embedded Workbench IDE.
4. *build**iar**pfc_interleaved_ccm_kv4x_iar.pmp*—the FreeMASTER project file for the application demonstration.
5. ..*src**common**rtcesl*\—this directory contains the RTCESL embedded libraries.
6. ..*src**common**sdk*\—this directory contains startup routines, linker files, header files, core-specific functions, and core-clock settings. These files are taken from the Kinetis SDK IDE. See www.nxp.com/KSDK for more information.
7. ..*src**projects**board**app_init.c*—the application peripheral initialization C source file.
8. ..*src**projects**board**app_init.h*—the application initialization header file.
9. ..*src**projects**board**clock_config.c*—the clock configuration C source file.
10. ..*src**projects**board**clock_config.h*—the clock configuration header file.
11. ..*src**projects**board**pfcdrv_hvp-kv46f.c*—the application peripheral initialization C source file for the specific board and MCU used.
12. ..*src**projects**board**pfcdrv_hvp-kv46f.h*—the application peripheral initialization header file.
13. ..*src**common**pfc_algorithms**pfc_control.c*—the PFC control algorithm C source file.
14. ..*src**common**pfc_algorithms**pfc_control.h*—the PFC control algorithm header file.
15. ..*src**common**pfc_drivers**pfcdrv_adc_adc12.c*—the PFC ADC driver C source file.
16. ..*src**common**pfc_drivers**pfcdrv_adc_adc12.h*—the PFC ADC driver header file.
17. ..*src**common**pfc_drivers**pfcdrv_pwma_pwm1ph.c*—the PFC PWM driver C source file.
18. ..*src**common**pfc_drivers**pfcdrv_pwma_pwm1ph.h*—the PFC PWM driver header file.
19. ..*src**projects**main.c*—the main C source file.
20. ..*src**projects**main.h*—the main header file.
21. ..*src**projects**pfc_appconfig.h*—the PFC configuration header file contains the definition of the constants for the application control, such as the parameters of the power stage, the PI controllers' parameters, the filters' parameters, and others.

NOTE

The software package is distributed as a stand-alone installation and all components are included in the project directory, so it is not needed to install any other components.

3.1.1. Data types

This application uses several data types: (un)signed integer, fractional, and accumulator. The integer data types are useful for general-purpose computation; they are familiar to the MPU and MCU programmers. The fractional data types enable the implementation of numeric and digital signal processing algorithms. The accumulator data type is a combination of both; that means it has the integer and fractional portions.

The following list shows the integer types defined in the libraries:

- Unsigned 16-bit integer—<0 ; 65535> with a minimum resolution of 1.
- Signed 16-bit integer—<-32768 ; 32767> with a minimum resolution of 1.
- Unsigned 32-bit integer—<0 ; 4294967295> with a minimum resolution of 1.
- Signed 32-bit integer—<-2147483648 ; 2147483647> with a minimum resolution of 1.

This list shows the fractional types defined in the libraries:

- Fixed-point 16-bit fractional—<-1 ; 1 - 2^{-15}> with a minimum resolution of 2^{-15}.
- Fixed-point 32-bit fractional—<-1 ; 1 - 2^{-31}> with a minimum resolution of 2^{-31}.

This list shows the accumulator types defined in the libraries:

- Fixed-point 16-bit accumulator—<-256.0 ; 256.0 - 2^{-7}> with a minimum resolution of 2^{-7}.
- Fixed-point 32-bit accumulator—<-65536.0 ; 65536.0 - 2^{-15}> with a minimum resolution of 2^{-15}.

3.2. Scaling of analog quantities

This equation shows the relationship between the real and fractional representations:

$$\boldsymbol{fractional\ value = \frac{Real\ value}{Real\ quantity\ range}} \qquad \textit{Eq. 1}$$

where:

- Fractional Value = the fractional representation of quantities [–]
- Real Value = the real quantity in physical units [..]
- Real Quantity Range = the maximum defined quantity value used for scaling in physical units [..]

Some examples of the quantities' scaling are provided in the following sub-sections.

3.2.1. Voltage scale

The voltage is generally measured on the voltage resistor divider by the ADC. Therefore, the maximum voltage scale is proportional to the maximum ADC input voltage range, and it is 443 V for the high-voltage power stage. The following equation shows how the fractional voltage variable is used:

Voltage scale: V_{max} = 443 V

Measured voltage: $V_{measured}$ = 390 V

$$\boldsymbol{(Frac16)voltage_{variable} = \frac{V_{measured}}{V_{max}} = \frac{390V}{443V} = 0.88036} \qquad \textit{Eq. 2}$$

This 16-bit fractional variable is stored internally as a 16-bit integer variable:

$$\boldsymbol{(Int16)voltage_{variable} = (Frac16)voltage_{variable}.2^{15} = 0.88036.2^{15} = 28848} \qquad \textit{Eq. 3}$$

Figure 3 illustrates the previous equations of the voltage scaling into a fractional number, the voltage value is read by the ADC as 12-bit signed number with a left justification for the 16-bit number.

In case the floating-point number representation of this value is needed, the fractional number is converted to the float number as:

$$(float)voltage_{variable} = \frac{V_{max}}{2^{15}}(Frac16)voltage_{variable} \quad \textit{Eq. 4}$$

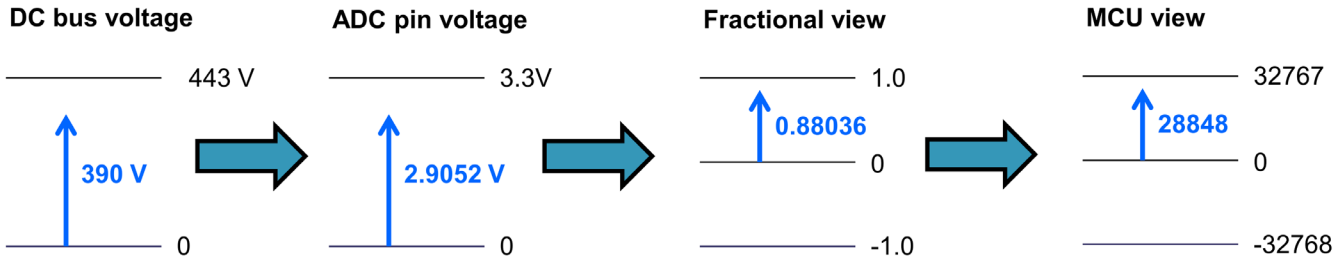

Figure 3. Voltage measurement

3.2.2. Current scale

The current is generally measured as a voltage drop on the PFC shunt resistors, which is amplified by an operational amplifier. Only the positive currents are measured on the shunt resistors. The amplified signal has an 82-mV offset due to the non-linearity of the operation amplifier. The maximum current scale is proportional to the maximum ADC input voltage range plus the offset (see Figure 4). The manipulation with the current variable is similar to the voltage variable manipulation.

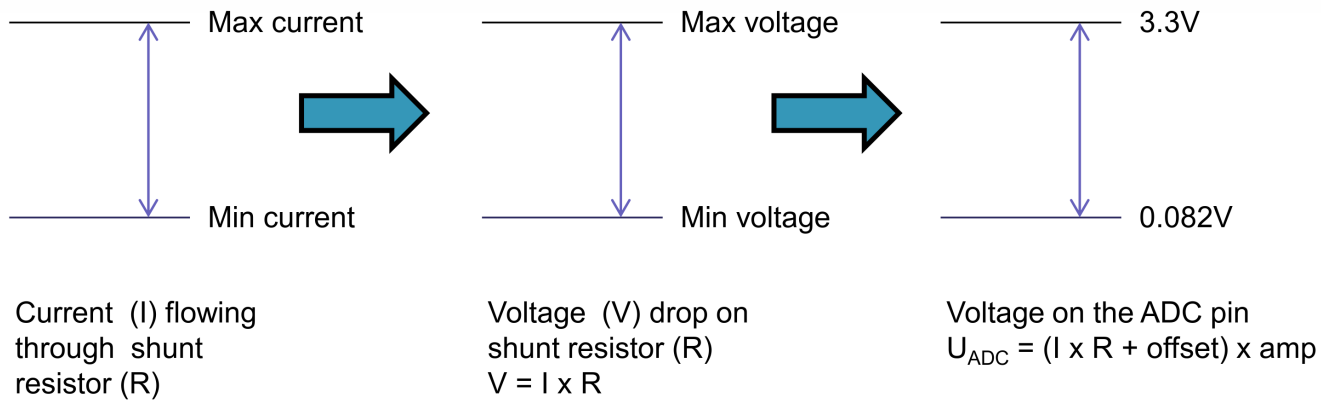

Figure 4. Current measurement

3.3. Application overview

The PFC software is interrupt-driven, with two PFC state machines running in two periodical interrupts. Besides the PFC state machines, there is the main application state machine running on the background in the main endless loop. The PFC control algorithm is designed to create one compact unit and can be easily incorporated into any target application. The entire PFC control algorithm is implemented in two Interrupt Service Routines (ISR). The two current loops are executed in the fast interrupt every 20 µs and the voltage loop is executed in the slow interrupt with a 1-ms period of execution. The ISR routines

are described in detail in Section 3.5, “PIT timeout interrupt” and Section 3.6, “Peripheral configuration”.

After an MCU reset, the application performs the clock and GPIO port peripheral initialization followed by the PFC peripheral drives initializion and the FreeMASTER embedded driver initialization. After a complete initialization, the global interrupt is enabled and the application jumps to the endless main loop. The background loop handles the time non-critical tasks, such as the application state machine and FreeMASTER communication. The flowchart of the main loop is shown in the following figure.

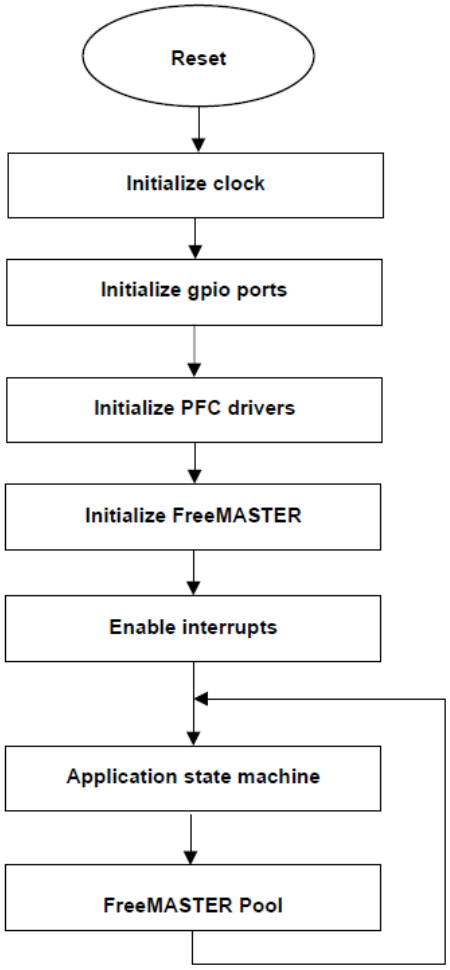

Figure 5. The main loop flowchart

The PFC control algorithm itself is driven by two PFC state machines called in the fast and slow control loops. The state of the fast PFC state machine is determined by the actual operational state of the PFC, while the slow state machine simply follows the fast one.

3.3.1. Application state machine

The PFC control algorithms can be easily integrated into other software. This reference code demonstrates the usage of the PFC control algorithm in a standalone PFC application. The application state machine controls the whole execution of the reference application. The application machine is

responsible for the proper PFC algorithm variables initialization and the control of the PFC algorithm according to the commands from the master system. In this reference application, the master system is represented by FreeMASTER, which enables you to control the PFC from a PC.

The application state machine consists of these four main states (see Figure 6):

- Fault—the application detects a fault condition and waits until the user clears the fault in FreeMASTER.
- Init—the application control variables initialization—the control and status words initialization.
- Stop—the application is initialized and waiting for the Run command from FreeMASTER. It can pass to the Fault state when a fault is detected.
- Run—the application is running. It may be stopped by the Stop command from FreeMASTER or pass to the Fault state when a fault is detected.

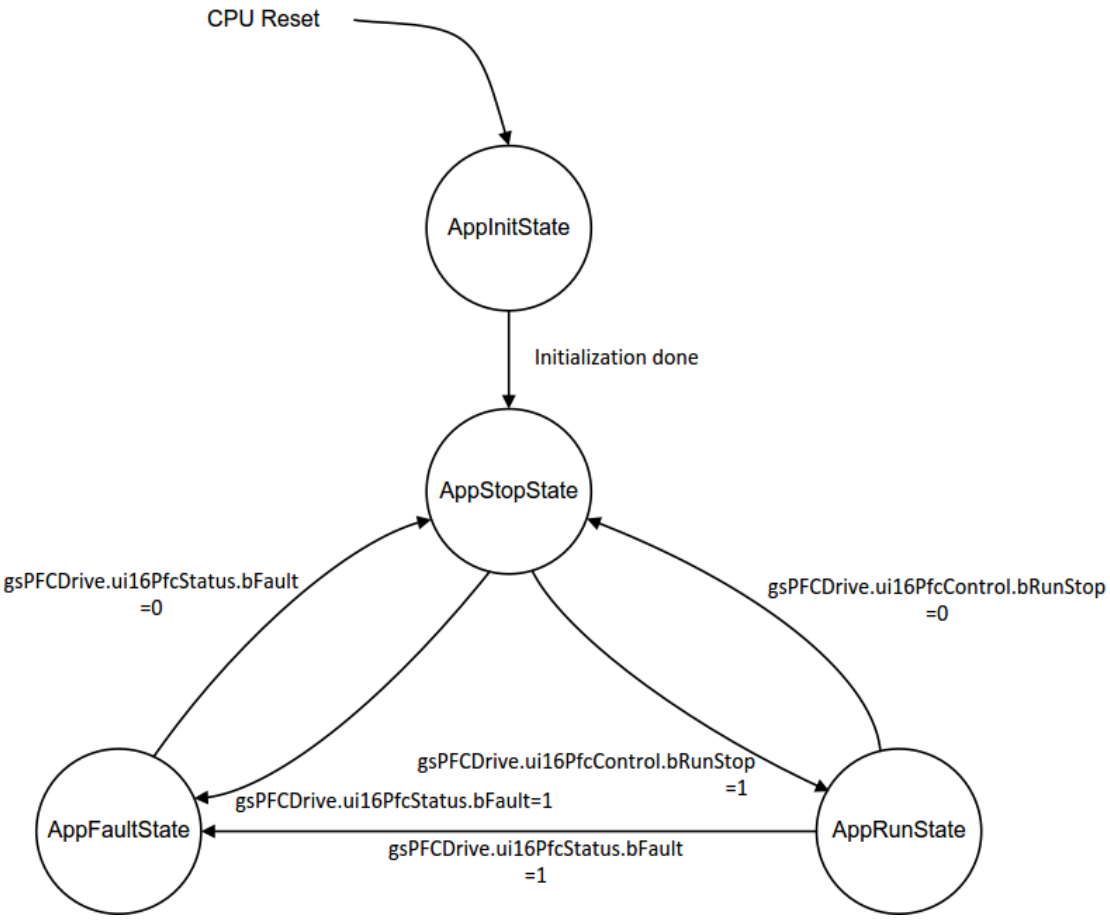

Figure 6. Application state machine

Each state in the state machine is represented by a particular function, which is executed periodically when the state machine is in the appropriate state. If the condition for a transition from one state to another is met, the transition function is called once during the transition. The transition function has a name of *xxToyy*, where *xx* stands for the initial state and *yy* stands for the target state. For example, during a transition from the Init state to the Stop state, the *InitToStop()* function is executed.

3.3.2. Fast PFC state machine

The fast PFC state machine *PFC_StateMachineFast* is shown in Figure 7. It is executed in the PWM ISR interrupt and incorporates these five states:

- *PfcFaultStateFast*—the PFC state machine stays in this state until a fault condition becomes valid. All faults are cleared after the preset time. If no fault condition persists, the state machine jumps to the Stop state.
- *PfcInitStateFast*—the PFC variables' initialization. When the initialization is done, the state machine jumps to the Stop state.
- *PfcStopStateFast*—the PFC is initialized and waiting for the Run command. It can pass to the Fault state when a fault is detected. The input voltage must be above the minimum value to stay in the Stop state and to not enter the Fault state.
- *PfcSoftStartStateFast*—the PFC soft start fast loop execution. It can be interrupted by the Stop command or pass to the Fault state when a fault is detected. When the nominal DC-Bus voltage is reached within a preset time, the state machine jumps to the Run state. Otherwise, it goes to the fault state.
- *PfcRunStateFast*—the PFC is running. It can be stopped by the Stop command or pass to the Fault state when a fault is detected.

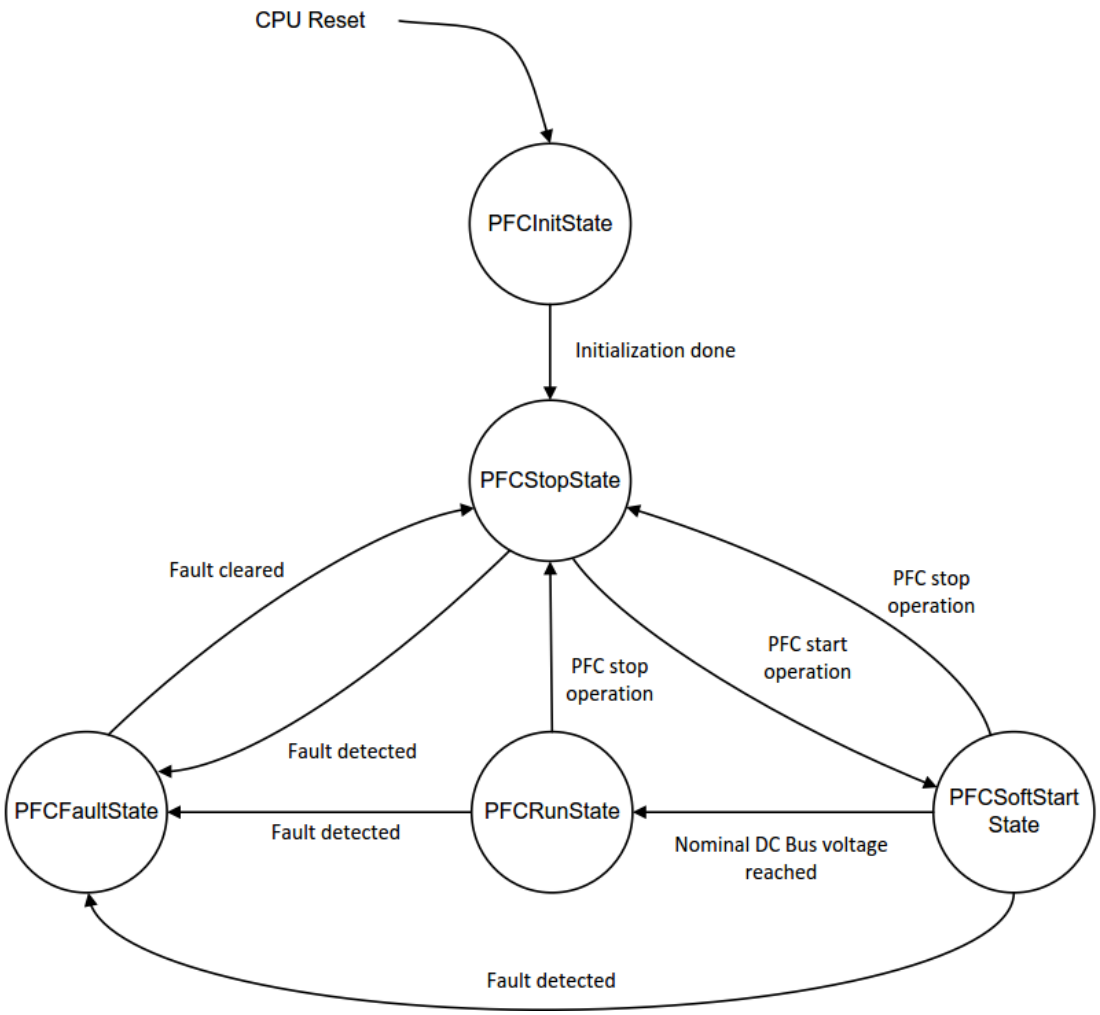

Figure 7. Fast PFC state machine

The transition functions are defined in the same way as for the application state machine.

3.3.3. Slow PFC state machine

The slow PFC state machine (shown in Figure 8) is executed in the PIT0 ISR and incorporates the same five states as the fast PFC state machine:

- *PfcFaultStateSlow*—an empty state.
- *PfcInitStateSlow*—an empty state.
- *PfcStopStateSlow*—an empty state.
- *PfcSoftStartStateSlow*—the PFC soft start slow loop execution. The DC-Bus voltage reference voltage is ramped up to the required DC-Bus voltage level.
- *PfcRunStateSlow*—the PFC run slow loop execution. The voltage controller keeps the DC-Bus voltage at the required level.

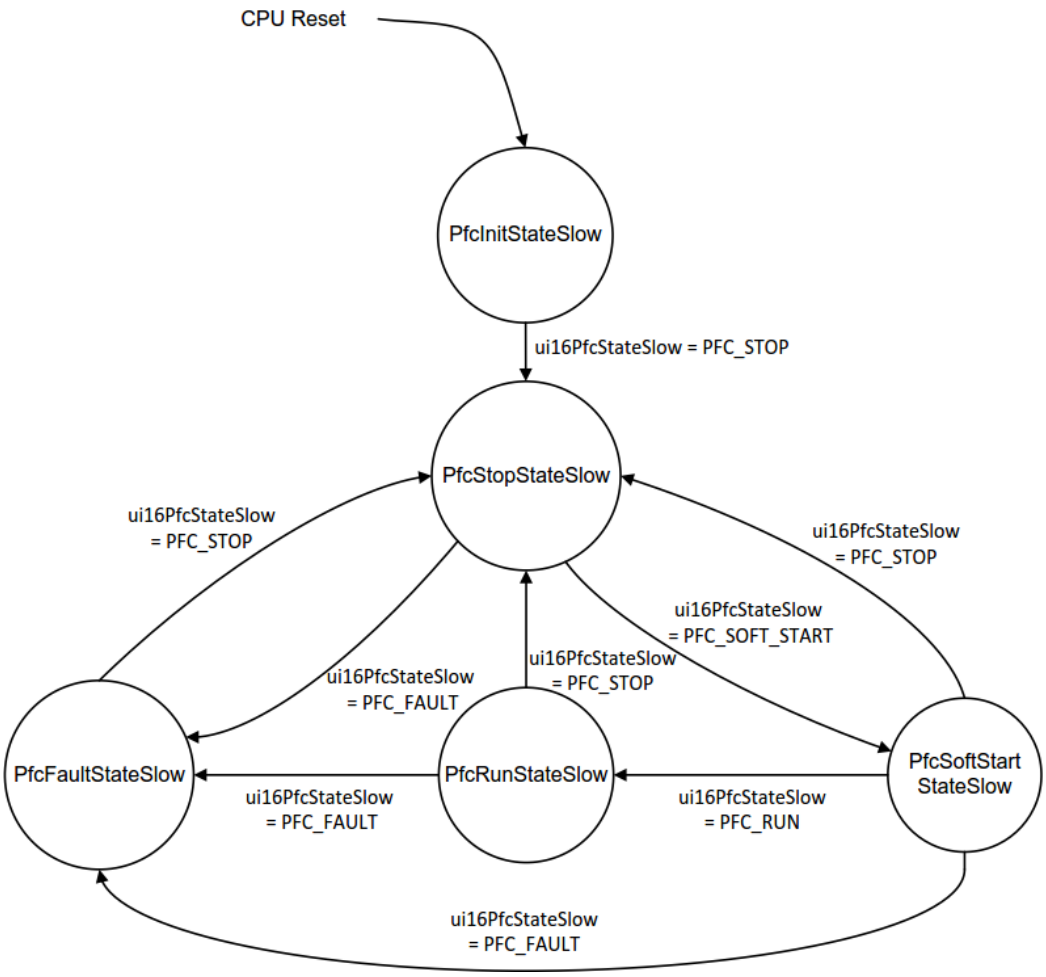

Figure 8. Slow PFC state machine

The transition functions are defined in the same way as for the application state machine.

3.4. PWM reload interrupt

The time-critical algorithms and the fast PFC state machine *PFC_StateMachineFast* are performed in the fast PWM interrupt service routine. The control algorithms implemented in the PWM ISR are:

- Fast PFC state machine.
- Read ADC quantities.
- Input voltage filter.
- DC-Bus voltage filter.
- PFC current filters.
- Current reference calculation—fast part.
- Current controllers.

The PWM ISR is executed regularly every 20 µs with the highest interrupt priority. The ADC conversion is synchronized with the PWM period and the conversion is started by two trigger signals. The trigger signal is always in the middle of the switching period for each phase. The complete application timing is described in Section 3.8, "Application timing". The ADC conversion runs in the background. The control algorithm execution depends on the actual state. The control algorithms executed in the particular states only are marked by a superscripted star (see Figure 9).

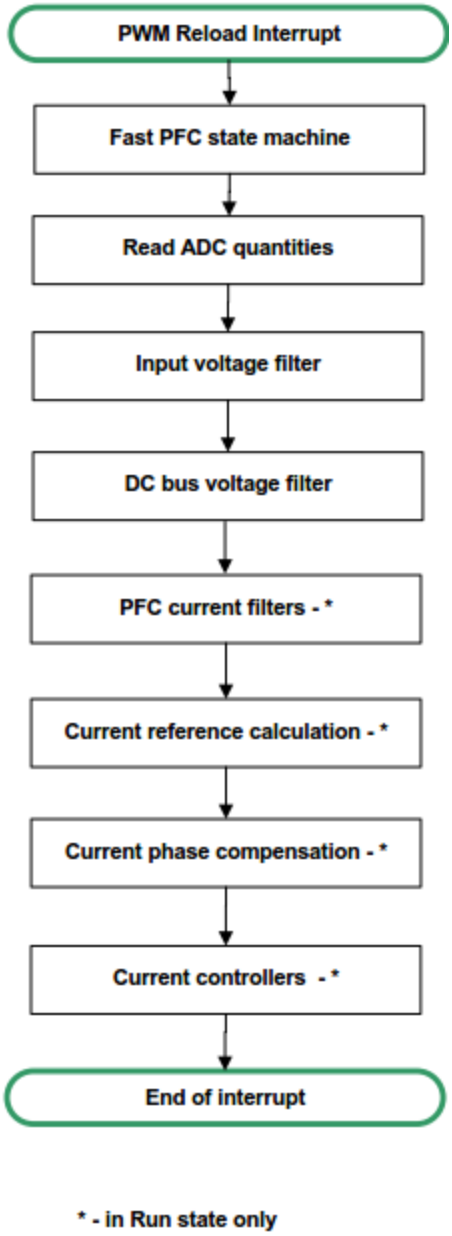

Figure 9. PWM ISR flow chart

3.5. PIT timeout interrupt

The PIT timeout interrupt is executed every 1 ms and has a lower priority level than the PWM interrupt. The slow PFC state machine *PFC_StateMachineSlow* is executed in this ISR. The control algorithms implemented in this interrupt are:

- Slow PFC state machine.
- Voltage controller.
- Current reference calculation—slow part.
- Power calculation.

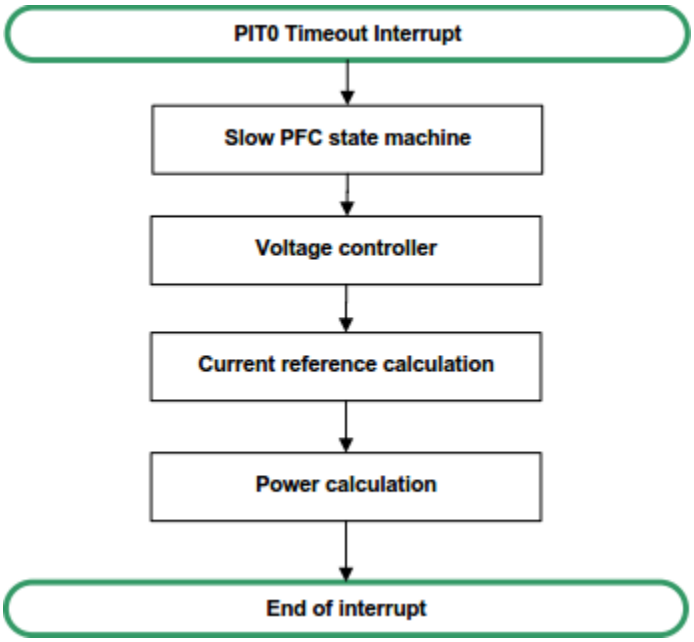

Figure 10. PIT0 ISR flow chart

3.6. Peripheral configuration

The application uses the dedicated peripherals for the PFC algorithm implementation and communication with FreeMASTER. The peripherals used in the application are: ADC, eFlexPWM, XBAR, PIT, and SCI. The other unused peripherals are disabled and not powered. The peripheral's initialization is described in the following subsections.

3.6.1. ADC (Analog to Digital Converter)

The ADC consists of two separate 12-bit ADCs with many analog inputs and its own S/H circuit that enable a fast conversion.

The ADC module is configured as follows:

- The IPBus clock/5 is the clock source—20 MHz.
- Single-ended channels configuration.

- Simultaneous parallel scanning.
- Trigger source from the eFlexPWM module (logic OR above Trigger 0 and Trigger 1).

3.6.2. eFlexPWM (Enhanced Flex Pulse Width Modulator)

The eFlexPWM is a dedicated peripheral that enables the generation of the control PWM signals. The PWM submodule 3 is used to generate two control signals for the boost converter MOSFET transistors. The PWM signal generation is shown in Figure 7.

The PWM submodule 3 is configured as follows:

- The IPBus clock is the clock source—100 MHz.
- Nano-edge PWM generation enabled.
- Independent, center-aligned operation mode.
- Modulo value 1000 ~ 100 kHz switching frequency.
- INIT value = - modulo value/2.
- VAL0, VAL2, VAL3 = 0.
- VAL1 = modulo value/2.
- VAL4 = - modulo value/2.
- VAL5 = modulo value/2.
- PWMA positive output polarity.
- PWMB negative output polarity.
- Full cycle reload and interrupt on every second PWM period (20 μs).
- Triggers 0 and 1 provide the synchronization signal to start the ADC conversion.

PWM Fault 0:

- The high-level detection indicates the over-current fault on the IPM module.
- Manual fault clearing.
- Fault input filter enabled.

3.6.3. XBAR A (crossbar switch module A)

The crossbar switch module A implements an array of 32 inputs and 40 outputs of the combinational digital multiplexes. This module provides a flexible connection from any input to any output under the user's control.

The application configuration is as follows:

- Channel 12, ADC Trigger:
  - The AOI EVENT0 signal is used for the ADC synchronization.
- Channel 14, PWM module Fault0:
  - The XBARIN7 (GPIOE.1) signal is routed to the PWM Fault 0 input.

3.6.4. XBAR B (crossbar switch module B)

The crossbar switch module B implements an array of 26 inputs and 16 outputs of the combinational digital multiplexes. This module provides a flexible connection from any input to any input of the And/Or/Inverter (AOI) module under the user's control.

The application configuration is as follows:

- The AOI EVENT0 input A is linked to Trigger 0 from the PWM submodule 3.
- The AOI EVENT0 input B is linked to Trigger 1 from the PWM submodule 3.

3.6.5. And/Or/Invert (AOI) module

The AOI module provides the boolean logic function operation among the on-chip peripheral control signals. The inputs' signals are linked over the XBAR B module. The logic OR function in the AOI EVENT 0 over the Trigger 0 and 1 signals from the PWM submodule 3 is used to generate the synchronization signal for the ADC module.

3.6.6. PIT timers

The PIT timers can generate trigger pulses and interrupts. Timer 0 is used to generate a periodic interrupt every 1 ms. The interrupt is used to execute the voltage control loop and calculate a part of the current reference. The PIT Timer 0 configuration is as follows:

- The IPBus clock is the clock source—25 MHz.
- The modulo value is set to 25000.
- The timeout interrupt is enabled.

3.6.7. SCI module

The SCI module is configured to communicate with the FreeMASTER application. The module is set to receive/transmit 8-bit data with a baud rate of 19200 and no parity.

3.7. PFC Control Peripheral Drivers (PFCDRV)

The PFCDRV provides a simple way of peripheral initialization and access for the PFC control. The features of the PFCDRV library include 2-phase PWM generation and 2-phase current measurement, as well as the measurement of the Input voltage, the DC-Bus voltage, and the IPM temperature (or one general user-defined auxiliary quantity).

The PFCDRV consists of these two parts:

- The first part is the peripheral initialization module consisting of the *pfcdrv_hvp-kv46f.c* and *pfcdrv_hvp-kv46f.h* files. These files are unique for each supported device. The header file includes all PFCDRV setup options, including the ADC channel assignment. The source file contains the functions to initialize all peripherals used for the PFC control. This module is described in Section 3.7.1, "PFCDRV initialization".
- The second part consists of the peripheral driver library modules for each supported periphery.

All ADC and PWM periphery drivers share the same API within their class. This enables the higher-level code to be platform-independent, because the peripheral driver function calls are replaced by the universally-named macros. The list of the supported peripherals and the APIs of their drivers are provided in Section 3.7.2, "PFCDRV API".

3.7.1. PFCDRV initialization

The PFCDRV initialization module consists of the MCU peripheral-initialization functions and all definitions. The functions are contained in the device-specific *pfcdrv_hvp-kv46f.c* source and *pfcdrv_hvp-kv46f.h* header files. From all the functions in the PFCDRV initialization module, it is only necessary to call the *PFCDRV_Init()* function during the MCU startup, before calling any other PFCDRV functions. All the peripherals used by the given device for the PFC control purposes are initialized within this function.

The *pfcdrv_hvp-kv46.h* header file provides several macros that you may define:

- PFCDRV_ADC—this macro specifies the ADC peripheral used. The KV4x family has a cyclic 12-bit ADC.
- PFCDRV_PWM1PH—this macro specifies the PWM peripheral used. The Pulse Width Modulator (PWM) A or the FlexTimer module can be selected. The PWM A was selected because it enables a high PWM generation with a 260-ps resolution.
- PFCDRV_TMR_SLOWLOOP—this macro specifies the timer used for the periodical slow loop execution. The PIT Timer 0 is selected.
- PFC_PWM_FREQ—the value of this definition sets the PWM frequency. Its value is used to calculate the modulo value.
- PFC_FREQ_VS_PWM_FREQ—the value of this definition represents the n-th PWM period, causing the PWM reload interrupt.
- PFC_SLOW_LOOP_FREQ—the value of this definition sets the slow loop execution period.
- PFC_PWM_PAIR—the value of this definition sets the PWM submodule to be used.

3.7.2. PFCDRV API

The ADC and PWM PFCDRV share the same API within their class. To ensure the device independency on the PFCDRV API, all driver functions are accessible through the universally-named macros in the *pfcdrv_hvp-kv46f.h* file. The available API of the ADC PFCDRV is:

- *PFCDRV_ADC_T*—PFCDRV ADC structure data type. The structure contains pointers that serve as a communication layer with the higher-level software. These variables are initialized by default in the *pfcdrv_hvp-kv46f.c* file in the initialization function of the selected ADC peripheral. The I/O layer variables are:
  - *pf16UDcBus*—pointer to a 16-bit fractional variable in which you want to store the measured DC-Bus voltage sample.
  - *pf16UInput*—pointer to a 16-bit fractional variable in which you want to store the measured input voltage sample.
  - *pf16I1ph*—pointer to a 16-bit fractional variable in which you want to store the measured boost current 1 sample.

  - *pf16I2ph*—pointer to a 16-bit fractional variable in which you want to store the measured boost current 2 sample.
- *bool_t PFCDRV_ADC_PERIPH_INIT()*—this function is by default called during the ADC peripheral initialization procedure triggered by the *PFCDRV_Init()* function, and it must not be called again after the peripheral initialization is done.
- *bool_t PFCDRV_ADC_GET(PFCDRV_ADC_T*)*—this function reads the measured quantities and stores their values in the related variables. This function always returns true.
- *bool_t PFCDRV_CURR_1PH_CALIB_INIT(PFCDRV_ADC_T*)*—this function initializes the boost current channel offset measurement. This function always returns true.
- *bool_t PFCDRV_CURR_1PH_CALIB(PFCDRV_ADC_T*)*—this function reads the current value from the unpowered boost converters and filters them using the moving average filters. The goal is to obtain the value of the measurement offset. The length of the window for the moving average filters is set by the *ui16TimeCalibration* variable. This function always returns true.
- *bool_t PFCDRV_CURR_1PH_CALIB_SET(PFCDRV_ADC_T*)*—this function asserts the boost current measurement offset values to the internal registers. Call it after the calibration time expires. This function always returns true.

The API for the PWM PFC-Control Peripheral Drivers is:

- *PFCDRV_PWM1PH_T*—PFCDRV PWM structure data type. The structure contains the I/O layer variable, which is initialized by default in the *pfcdrv_hvp-kv46f.c* file in the initialization function of the selected PWM periphery. The I/O layer variables are:
  - *pf16Duty1*—pointer to a 16-bit fractional variable in which the duty cycle for the boost converter 1 is stored.
  - *pf16Duty2*—pointer to a 16-bit fractional variable in which the duty cycle for the boost converter 2 is stored.
- *bool_t PFCDRV_PWM_PERIPH_INIT(PFCDRV_PWM1PH_T *)*—this function is called by default during the PWM periphery-initialization procedure triggered by the *PFCDRV_Init()* function. This function always returns true.
- *bool_t PFCDRV_PWM1PH_SET(PFCDRV_PWM1PH_T *)*—this function updates the PWM duty cycles based on the values stored in the *f16Duty1* and *f16Duty2* variables. This function always returns true.
- *bool_t PFCDRV_PWM1PH_EN(PFCDRV_PWM1PH_T*)*—calling this function enables all PWM channels. This function always returns true.
- *bool_t PFCDRV_PWM1PH_DIS(PFCDRV_PWM1PH_T *)*—calling this function disables all PWM channels. This function always returns true.
- *bool_t PFCDRV_PWM1PH_FLT_GET(PFCDRV_PWM1PH_T *)*—this function returns the state of the over-current fault flag and automatically clears the flags (if set). This function returns true when the over-current events occur. Otherwise, it returns false.

3.8. Application timing

This section describes the PFC control application timing. The graphical representation of the application timing is shown in Figure 11. All the tasks are handled in the interrupt service routines or

within the main loop. The fast control loop is executed in the *PWMA_RELOAD3_IRQHander()* interrupt. The slow control loop is executed in the *PIT0_IRQHander()* interrupt. The FreeMASTER pool function *FMSTR_Pool()* is handled in the infinite main loop.

The application timing is based on the PWM generation with a timebase of 10 µs. The fastest interrupt is executed regularly every second PWM period (20 µs) within the highest interrupt priority. The slower interrupt is executed regularly every 1 ms with a lower interrupt priority. The remaining time is filled by the FreeMASTER communication with the host PC and the application state machine. The ADC conversion is synchronized with the PWM period and the conversion is started by two trigger signals. One trigger signal is in the middle of the switching period defined by the VAL0 register. The analog quantities are sampled in this order: the input voltage, the DC-Bus voltage, and the boost current 1. The second trigger signal is at the end of the switching period (the middle of the second phase) defined by the VAL1 register. At this moment, the boost current 2 and the IPM module temperature are sampled. Because the boost currents are measured in the middle of the switching periods, the current samples represent the average currents of each boost converter leg. The ADC conversion takes up to 780 ns. The ADC conversion time is considered during the fast interrupt execution. The ADC conversion must be finished before the current controllers are calculated.

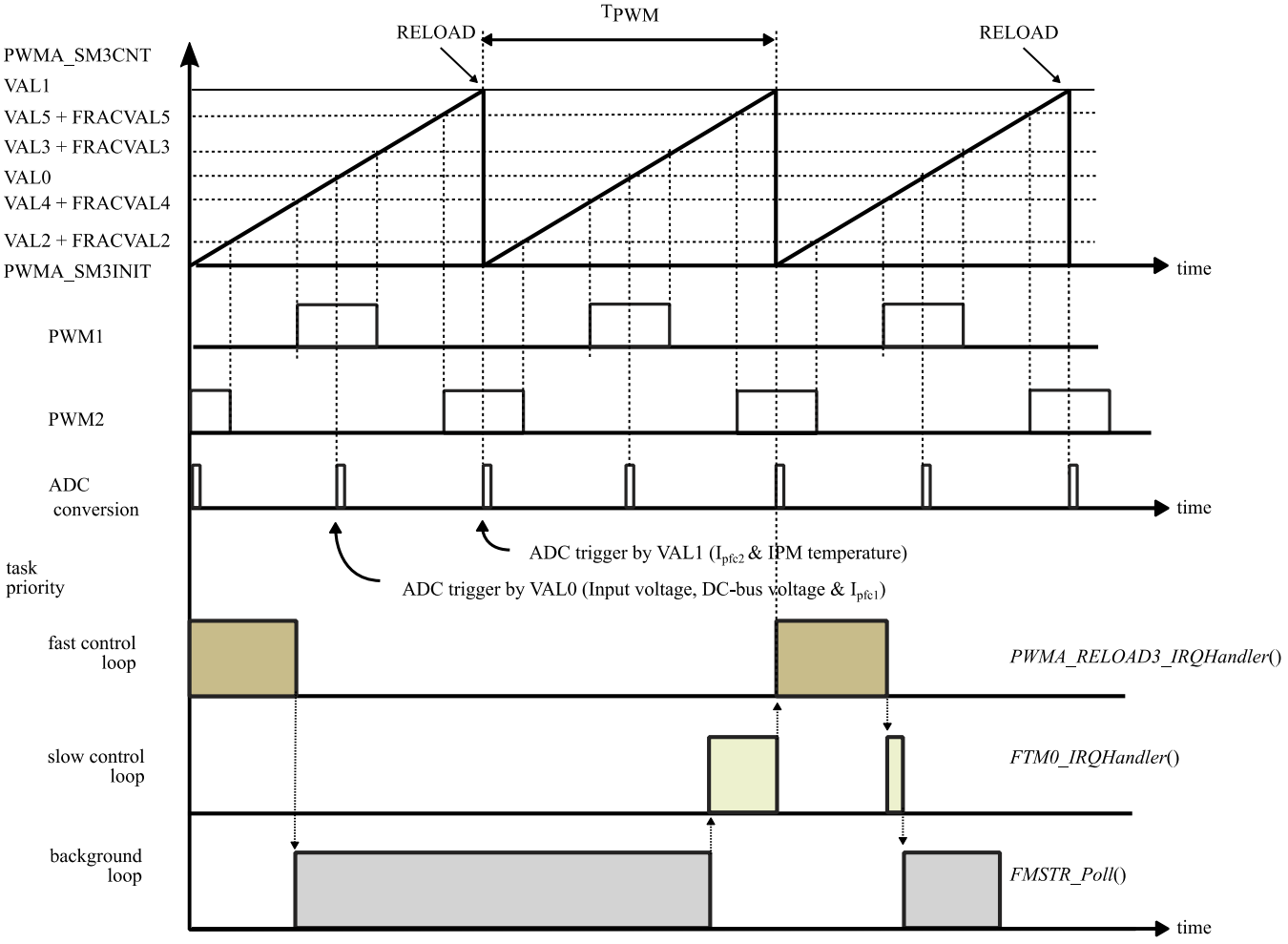

Figure 11. Application timing

4. Control loops design

The control loops design comes from the small signal model of the boost converter. The mathematical model of this converter and the control theory is described in the *Average Current Mode Interleaved PFC Control* (document AN5257). All equations references mentioned in this section correspond to the equations references in the *Average Current Mode Interleaved PFC Control* (document AN5257). The average current control mode of the boost converter control is implemented using three control loops/blocks: the current loop, the voltage loop, and the feed-forward block. The current controller, the voltage controller, and the low-pass filter for the feed-forward block are described in this section. The parameters of the boost converter for the control loops design are summarized in the following table.

Table 1. Boost converter parameters

Parameter	Value
Output power per leg	400 W
Boost inductor	650 µH
DC-Bus capacitor	660 µF
Switching frequency	100 kHz
DC-Bus voltage	400 V
Input voltage	230 VAC
Sampling frequency	50 kHz

4.1. Current controller design

The transfer function of the power stage in the current loop with the parameters from Table 1 is given by this equation:

$$G_{id}(s) = \frac{V_{at}}{sL} = \frac{400}{s \cdot 6.5 \cdot 10^{-4}} = \frac{6.1538 \cdot 10^{5}}{s} \qquad \textbf{Eq. 5}$$

T_D represents the system delay caused by the digital control. The delay is the sum of the sampling delay and the digital PWM modulator delay calculated according to Eq. 7. The total delay is 15 µs. The delay in the system is represented by the first-order Pade approximation given by Eq. 6.

$$G_{ID}(s) = \frac{1 - \frac{s \cdot 1.5 \cdot 10^{-5}}{2}}{1 + \frac{s \cdot 1.5 \cdot 10^{-5}}{2}} = \frac{1 - s \cdot 7.5 \cdot 10^{-6}}{1 + s \cdot 7.5 \cdot 10^{-6}} \qquad \textbf{Eq. 6}$$

$$T_D = \frac{20 \cdot 10^{-6}}{2} + \frac{10 \cdot 10^{-6}}{2} = 1.5 \cdot 10^{-5} \qquad \textbf{Eq. 7}$$

The bode plot of the power stage transfer function (including the digital delay) is shown in the following figure.

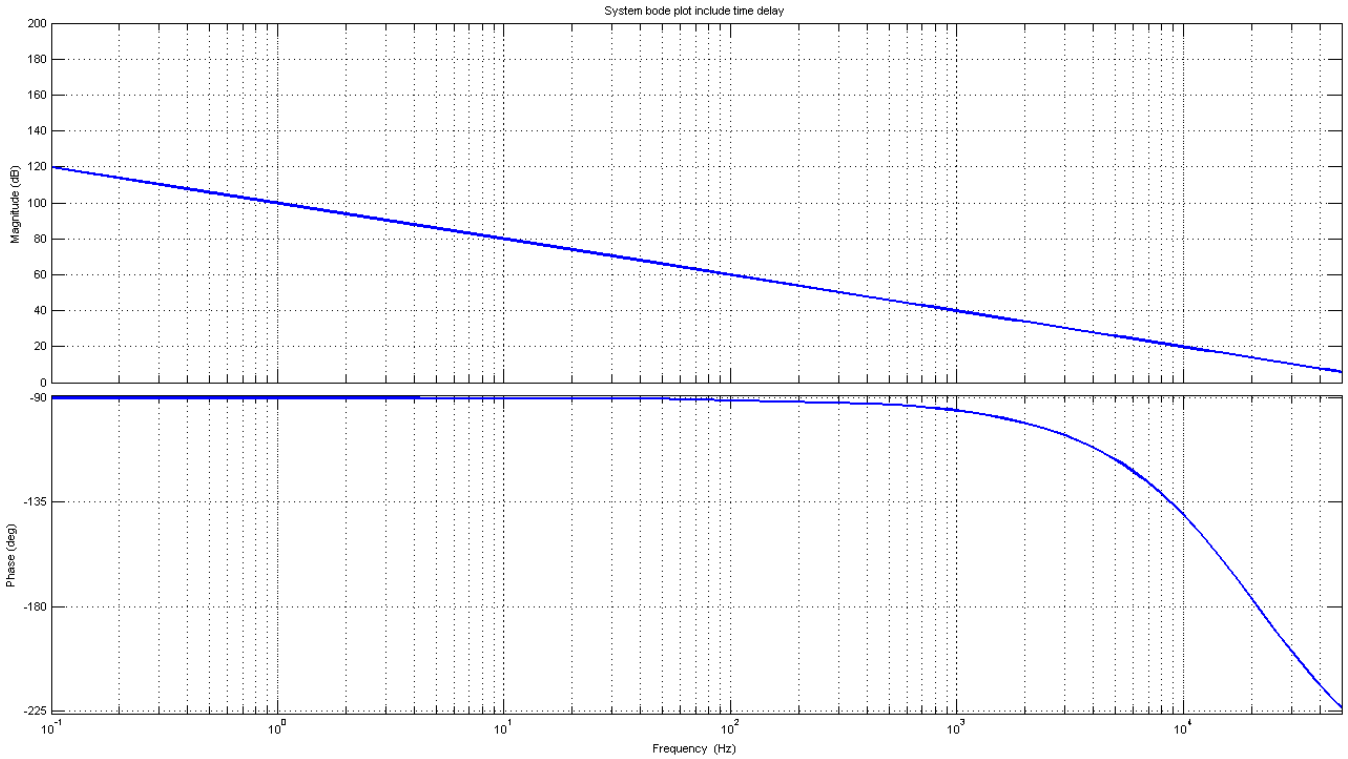

Figure 12. Bode plot of power stage including digital delay

The block diagram of the inner current control loop is shown in this figure:

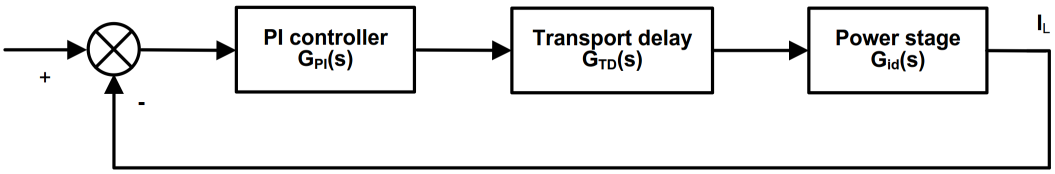

Figure 13. Current loop

$G_{PI}(s)$ represents the transfer function of the PI controller. $G_{id}(s)$ represents the transfer function of the current-control loop. For the current loop controller design, the bandwidth of the control loop is set to 4 kHz and the phase margin is set to 65°. To meet the accurate tracking of the current reference signal and achieve a good dynamic performance, the current controller must have a wide bandwidth and high gain at low frequencies and a good phase margin. The open loop transfer function of the current loop is shown in this equation:

$$G_{Open}(s) = \left(65.3535 \cdot \frac{1 + \frac{s}{1.6035 \cdot 10^3}}{s} \right) \cdot \frac{1 - s \cdot 7.5 \cdot 10^{-6}}{1 + s \cdot 7.5 \cdot 10^{-6}} \cdot \frac{6.1538 \cdot 10^5}{s} = \frac{-25.08 \cdot 10^3 \cdot (s - 1.333 \cdot 10^5) \cdot (s + 1603)}{s^2 \cdot (s + 1.333 \cdot 10^5)} \quad \textit{Eq. 8}$$

Firstly, the controller design is started by the calculation of the controller zero from the phase characteristics. The phase margin is defined at a cross-over frequency and the controller zero is calculated from the phase characteristics using the following equation.

$$\omega_z = \frac{2 \cdot \pi \cdot 4 \cdot 10^3}{\tan(65° + 2\arctan(\frac{1.5 \cdot 10^{-5}}{2} \cdot 2 \cdot \pi \cdot 4 \cdot 10^3)} = 1.6035 \cdot 10^3 \, rad / s$$ **Eq. 9**

The controller integral gain is calculated in the second step. The integral gain is calculated from the gain characteristic at a cross-over frequency using this equation:

$$K_I = \frac{6.5 \cdot 10^{-4}}{400} \cdot \frac{(2 \cdot \pi \cdot 4 \cdot 10^3)^2}{\sqrt{1 + (\frac{2 \cdot \pi \cdot 4 \cdot 10^3}{1.6035 \cdot 10^3})^2}} = 65.3535$$ **Eq. 10**

Finally, the proportional gain of the controller is calculated using this equation:

$$K_P = \frac{65.3535}{1.6035 \cdot 10^3} = 0.0408$$ **Eq. 11**

The bode plot of the open control loop in the following figure shows that the control system has the required control-loop bandwidth of 4 kHz and the phase margin of 65°.

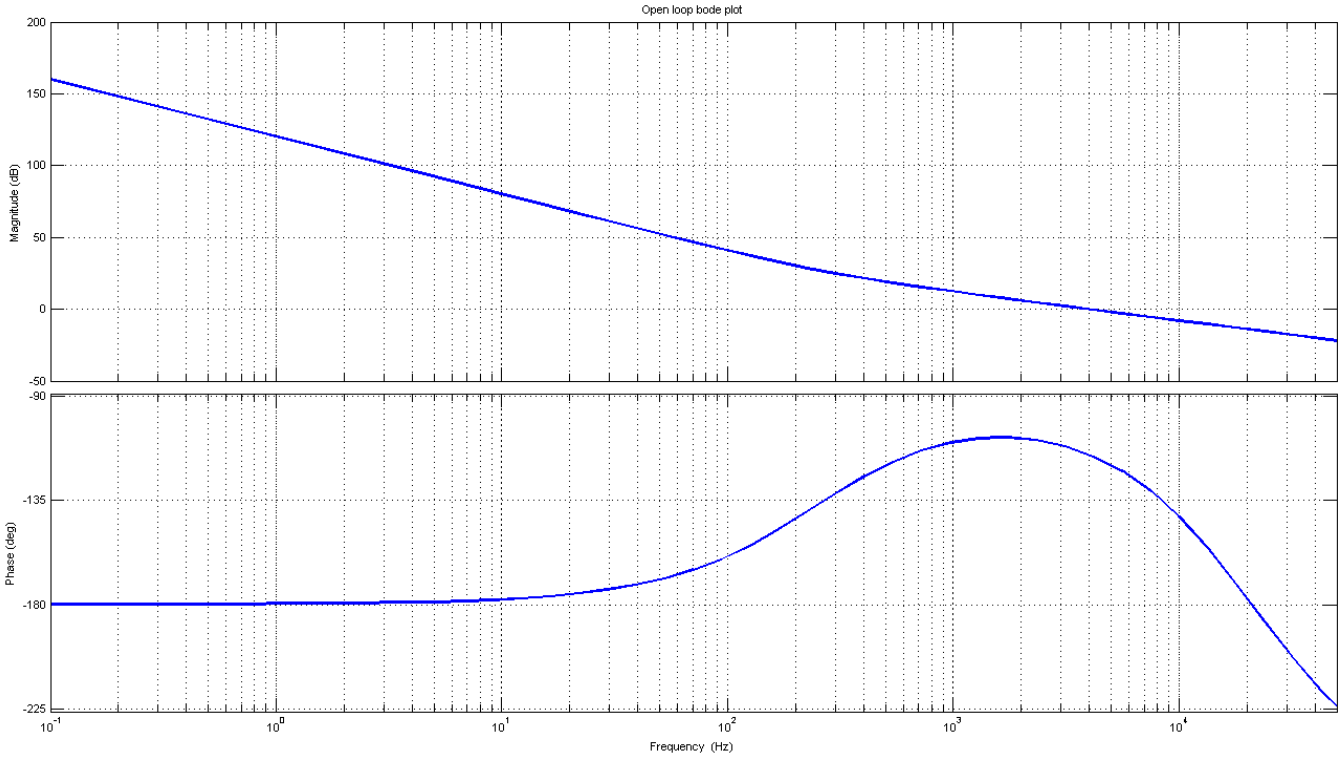

Figure 14. Open current loop bode plot

The proportional and integral gains of the PI controller are scaled to the ACC32 format using Eq. 12 for the proportional gain and Eq. 13 for the integral gain.

$$G_{P\_scaled} = 0.0408 \cdot \frac{8.0}{1.0} = 0.326 \quad \textit{Eq. 12}$$

$$G_{I\_scaled} = 65.024 \cdot 2 \cdot 10^{-5} \cdot \frac{8.0}{1.0} \cdot = 0.0105 \quad \textit{Eq. 13}$$

The scaled current controller parameters are $G_{P\_scaled} = 0.326$ and $G_{I\_scaled} = 0.0105$. The calculated constants are defined in the *pfc_appconfig.h* file as follows:

```
//Current Controller - Parallel type
#define I_KP_GAIN                       ACC32(0.326)
#define I_KI_GAIN                       ACC32(0.0105)
```

4.2. Voltage controller design

The transfer function of the power stage in the voltage loop is given by the following equation from *Average Current Mode Interleaved PFC Control* (document AN5257) using the design parameters from Table 1. The processing delay is ignored due to the oversampling.

$$G_{vi}(s) = \frac{\sqrt{2} \cdot 230 \cdot \frac{400}{2}}{2 \cdot 400 \cdot \left(\frac{400}{2} \cdot 6.6 \cdot 10^{-4} s + 1 \right)} = \frac{616.04}{s + 7.576} \quad \textit{Eq. 14}$$

The outer voltage control loop is described in the following figure. G_{PI} represents the gain of the PI controller and G_{vi} represents the power stage transfer function with the closed current loop gain.

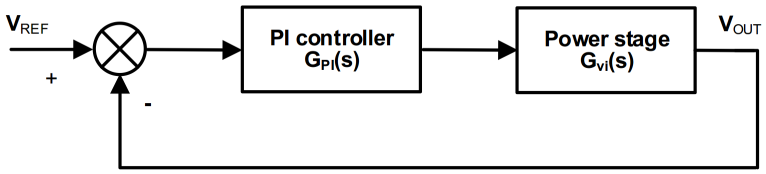

Figure 15. Voltage loop

The transfer function of the voltage open loop is described by this equation:

$$G_{Open}(s) = 0.7727 \cdot \left(\frac{1 + \frac{s}{7.576}}{s}\right) \cdot \frac{616.04}{s + 7.576} = \frac{62.832 \cdot (s + 7.576)}{s \cdot (s + 7.576)} \quad \textit{Eq. 15}$$

The design of the voltage controller is calculated according to the required bandwidth ω_c and the phase margin φ_{pm} of the voltage control loop. Because the bandwidth of the voltage control loop is significantly smaller than the bandwith of the current control loop, the current control loop is replaced by the unity gain in the voltage open loop transfer function. The phase margin is set to 90° because the voltage overshoot is not wanted and the bandwidth of the control loop must be set between 1/10 to 2/10 of $2\omega_s$ to sufficiently filter the double grid frequency ($2\omega_s$). The bode plot of the voltage open loop is shown in the following figure.

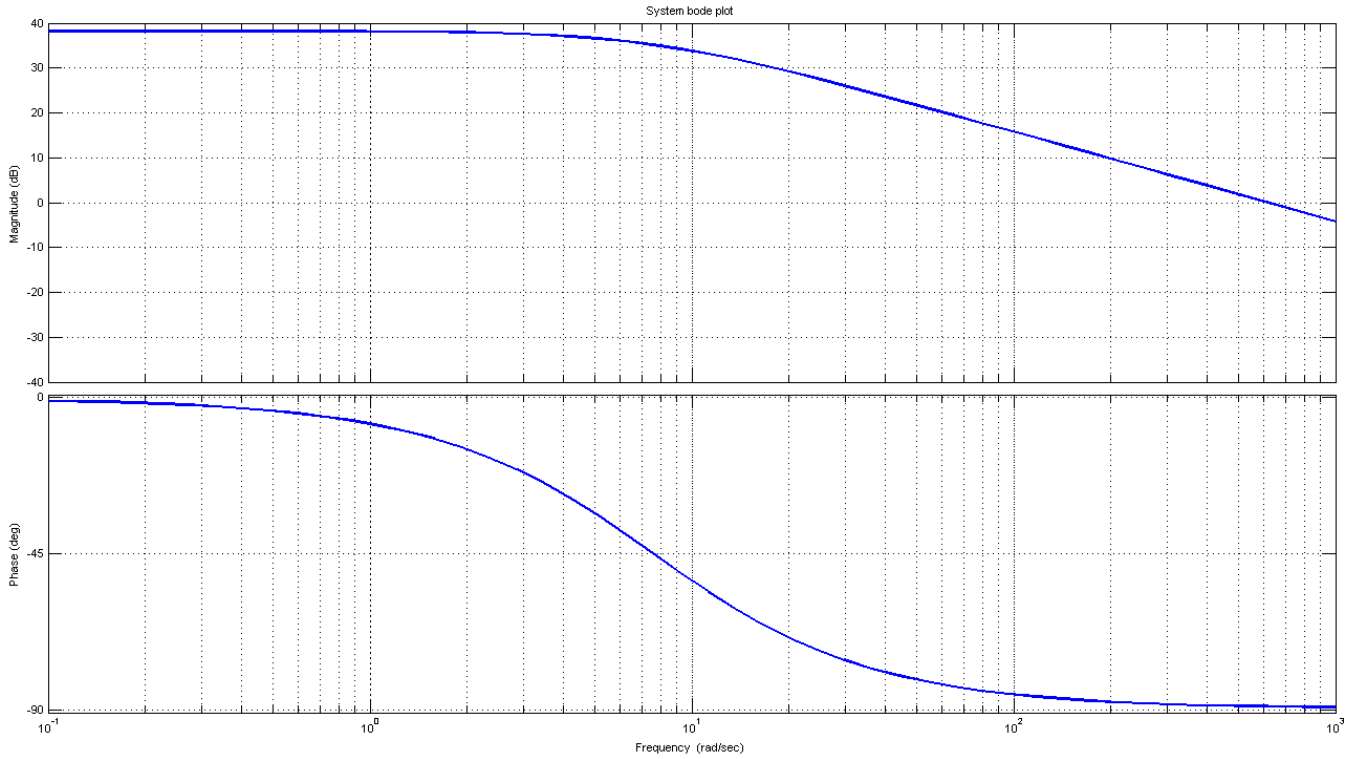

Figure 16. Voltage loop system bode plot

The voltage controller parameters are calculated in the same two steps as the current controller. Firstly, the controller zero from the phase characteristics must be calculated. The phase margin is defined at the cross-over frequency and the controller zero is calculated using this equation:

$$\omega_z = \frac{2 \cdot \pi \cdot 10}{\tan(-\frac{\pi}{2} + 90° + \arctan(2 \cdot \pi \cdot 10 \cdot 6.6 \cdot 10^{-4} \cdot \frac{400}{2}))} = 7.576 rad/s \qquad \textit{Eq. 16}$$

The integral gain of the controller is calculated in the second step. The integral gain is calculated from the gain characteristic at the cross-over frequency using this equation:

$$K_I = \frac{4V_{OUT}}{R\overline{V}_{IN}} \frac{\omega_c \sqrt{1+(\omega_c C \frac{R}{2})^2}}{\sqrt{1+(\frac{\omega_c}{\omega_z})^2}} = \frac{4 \cdot 400}{400 \cdot \sqrt{2} \cdot 230} \frac{2 \cdot \pi \cdot 10 \cdot \sqrt{1+(2 \cdot \pi \cdot 10 \cdot 6.6 \cdot 10^{-4} \frac{400}{2})^2}}{\sqrt{1+(\frac{2 \cdot \pi \cdot 10}{50.095})^2}} = 0.7727 \qquad \textit{Eq. 17}$$

Finally, the proportional gain of the controller is calculated using this equation:

$$K_P = \frac{0.7727}{7.576} = 0.102 \qquad \textit{Eq. 18}$$

The bode plot of the open control loop in the following figure shows that the control system has the required control loop bandwidth of 10 Hz and the phase margin of 90°. The proportional and integral

controller gains are further scaled to the ACC32 digital format using Eq. 12 for the proportional gain and Eq. 13 for the integral gain.

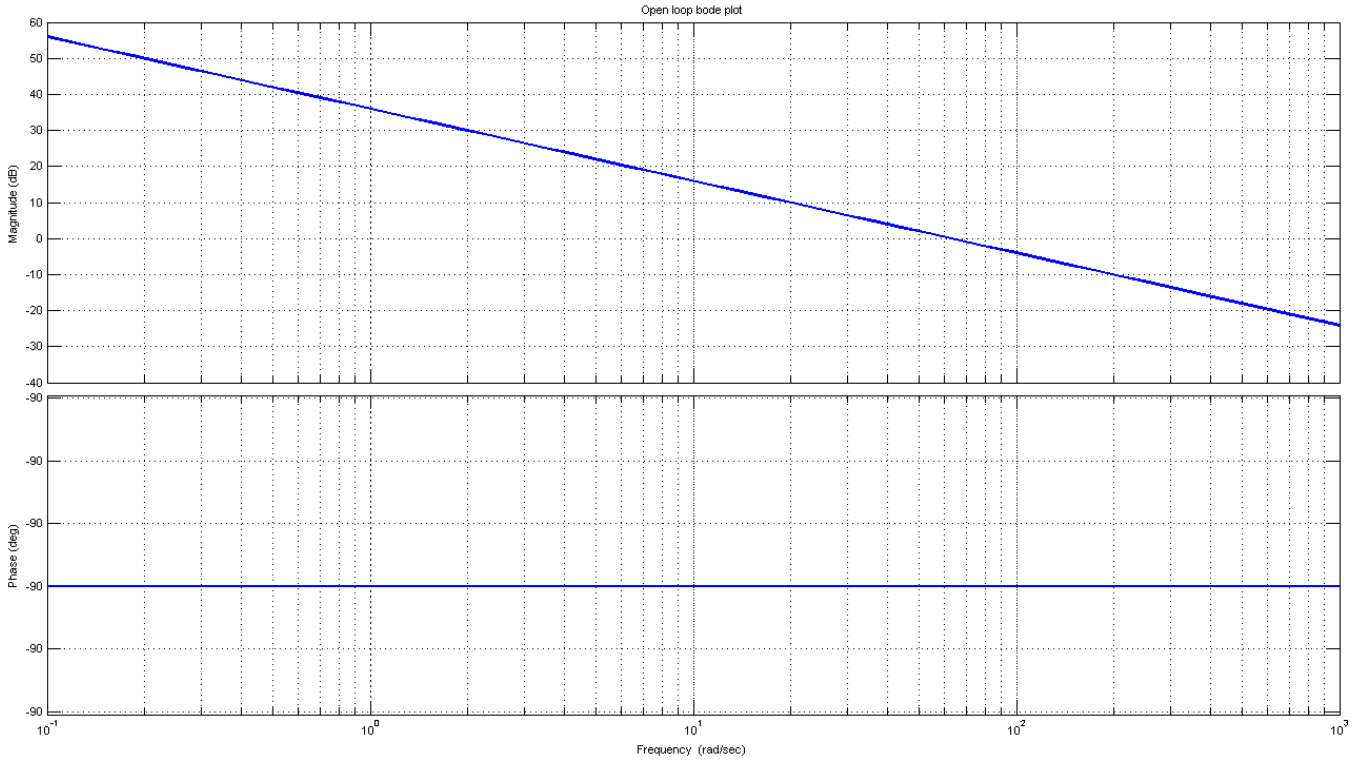

Figure 17. Open voltage loop bode plot

The scaled voltage controller parameters are $G_{P\_scaled} = 5.648$ and $G_{I\_scaled} = 0.0428$. The calculated constants are defined in the *pfc_appconfig.h* file as follows:

```
//Voltage Controller - Parallel type
#define U_KP_GAIN                       ACC32(5.648)
#define U_KI_GAIN                       ACC32(0.0428)
```

4.3. Low-pass filter design

The input voltage RMS value is needed for the voltage feed-forward block. The RMS value is obtained by filtering the input voltage using the second-order (n=2) Butterworth filter. This is realized by the second-order IIR filter *GDFLIB_FilterIIR2*, taken from the RTCESL general digital function library (see the *Set of General Digital Filters for Cortex M4 Core User's Guide* (document CM4GDFLIBUG)). The transfer function for the second-order Butterworth filter in the analog domain is shown in this equation:

$$H(s) = \frac{\omega_c^{\,2}}{s^2 + s\sqrt{2}\omega_c + \omega_c^{\,2}} \qquad \textbf{Eq. 19}$$

The stop frequency filter f_s is set to 100 Hz. The maximum ripple of the filtered voltage is calculated from the required THD (Total Harmonic Distortion). If the THD is set to 1.5 %, calculate the gain using this equation:

$$G_S = 20\log(0.015) = -36.48dB \quad \textbf{Eq. 20}$$

The cut-off frequency is calculated using this equation:

$$\omega_c = \frac{\omega_{STOP}}{\sqrt[n^2]{10^{-G_s/10}-1}} = \frac{2\pi 100}{\sqrt[4]{10^{3.648}-1}} = 76.95rad \cdot s^{-1} \quad \textbf{Eq. 21}$$

The filter coefficients of the Butterworth filter (a_1, a_2, b_0, b_1, and b_2) are calculated using the following equations, where T is the execution period of 20 µs (50 kHz).

$$b_0 = \frac{\omega_c^2}{\frac{4}{T^2}+\frac{2\sqrt{2}}{T}\omega_c+\omega_c^2} = \frac{76.95^2}{\frac{4}{(2\cdot 10^{-5})^2}+\frac{2\sqrt{2}}{2\cdot 10^{-5}}\cdot 76.95+76.95^2} = 6.57101\cdot 10^{-7} \quad \textbf{Eq. 22}$$

$$b_1 = \frac{2\omega_c^2}{\frac{4}{T^2}+\frac{2\sqrt{2}}{T}\omega_c+\omega_c^2} = \frac{2\cdot 76.95^2}{\frac{4}{(2\cdot 10^{-5})^2}+\frac{2\sqrt{2}}{2\cdot 10^{-5}}\cdot 76.95+76.95^2} = 1.3142\cdot 10^{-6} \quad \textbf{Eq. 23}$$

$$b_2 = \frac{\omega_c^2}{\frac{4}{T^2}+\frac{2\sqrt{2}}{T}\omega_c+\omega_c^2} = \frac{76.95^2}{\frac{4}{(2\cdot 10^{-5})^2}+\frac{2\sqrt{2}}{2\cdot 10^{-5}}\cdot 76.95+76.95^2} = 6.57101\cdot 10^{-7} \quad \textbf{Eq. 24}$$

$$a_1 = \frac{2\omega_c^2-\frac{8}{T^2}}{\frac{4}{T^2}+\frac{2\sqrt{2}}{T}\omega_c+\omega_c^2} = \frac{2\cdot 76.95^2-\frac{8}{(2\cdot 10^{-5})^2}}{\frac{4}{(2\cdot 10^{-5})^2}+\frac{2\sqrt{2}}{2\cdot 10^{-5}}\cdot 76.95+76.95^2} = -1.99782332 \quad \textbf{Eq. 25}$$

$$a_2 = \frac{\frac{4}{T^2}-\frac{2\sqrt{2}}{T}\omega_c+\omega_c^2}{\frac{4}{T^2}+\frac{2\sqrt{2}}{T}\omega_c+\omega_c^2} = \frac{\frac{4}{(2\cdot 10^{-5})^2}-\frac{2\sqrt{2}}{2\cdot 10^{-5}}\cdot 76.95+76.95^2}{\frac{4}{(2\cdot 10^{-5})^2}+\frac{2\sqrt{2}}{2\cdot 10^{-5}}\cdot 76.95+76.95^2} = 0.997825686 \quad \textbf{Eq. 26}$$

Because the second-order filter returns the average value of the input signal, all the b-coefficients of the filter are multiplied by 1.110721 ($\pi/2\sqrt{2}$) to obtain the RMS value. The calculated constants are defined in the *pfc_appconfig.h* file as follows:

```
// Input Voltage 2nd order LP filter constants
#define UIN_IIR_B0                      FRAC32(0.000000657101/2)
#define UIN_IIR_B1                      FRAC32(0.0000013142/2)
```

```
#define UIN_IIR_B2                      FRAC32(0.000000657101/2)
#define UIN_IIR_A1                      FRAC32(-1.99782332/-2)
#define UIN_IIR_A2                      FRAC32(0.997825686/-2)
```

All the filter constants in the algorithm are divided by two, as described in the *Set of General Digital Filters for Cortex M4 Core User's Guide* (document CM4GDFLIBUG).

5. Application setup and control

This chapter describes the setup of the demo application, building and loading the demo code into the MCU, and the user control of the PFC using FreeMASTER.

5.1. Hardware setup

The PFC application is built using the High Voltage Motor Control Platform (HVP-MC3PH) with the MKV46F150M MCU. The complete hardware setup is shown in this figure:

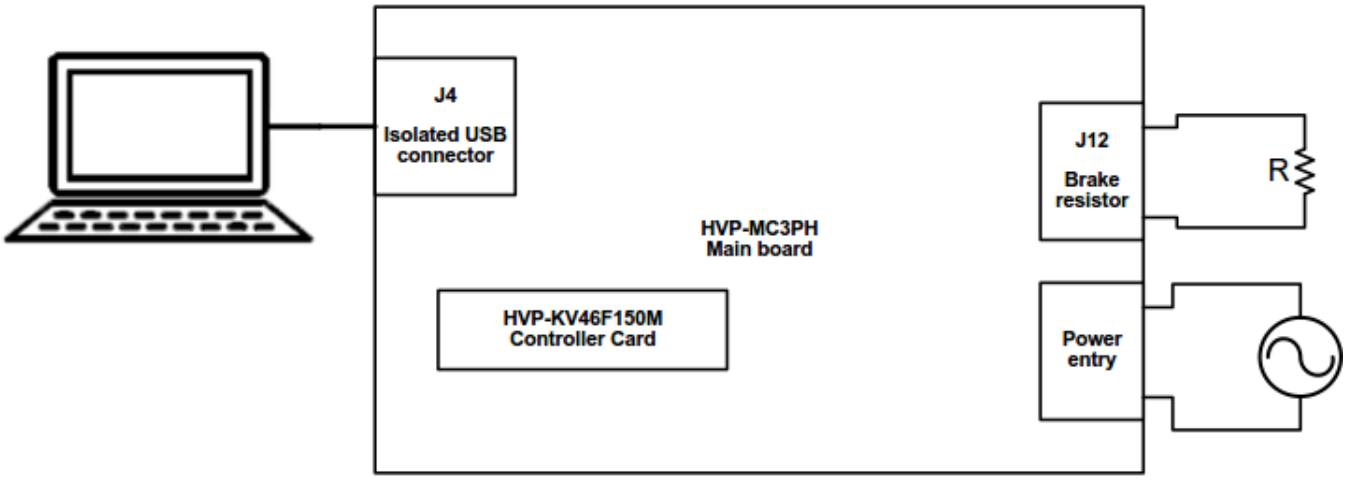

Figure 18. Hardware setup

Both boards are ready for the PFC development in their default configuration, so no jumper setting is needed before running the application. The load is connected through the brake resistor connector J12. The high-voltage electronic load in the constant resistance mode is preffered. However, any 200 Ω (or larger) resistor can be used. The power rating of the resistor must be higher than 800 W. The high-voltage power stage can be supplied directly from the mains. The supply voltage range is 90-240 VAC.

5.2. Building and debugging application

The software package contains the project files for the IAR Embedded Workbench IDE. The software package is distributed as a standalone installation, so all the components are included in the project directory and it is not needed to install any other components.

The project can be opened by double-clicking the workspace data file *hvp_kv46_pfc_ccm_interleaved.eww*. The point number 1 in Figure 19 shows the IAR workspace with

an opened project. The project opened in the IAR Embedded Workbench IDE is fully configured and includes all the source and header files required by the application, such as startup code, clock configuration, and peripheral configuration. Choose one of the two compiling conditions ("debug" or "release") shown in Figure 19, point 2. Each of the two conditions has its own setting:

- Debug—used for debugging, the optimization has the "None—turned off" flag.
- Release—used for releasing, the optimization has the "High—highest optimization for speed" flag.

The source code shown in Figure 19 includes these source files and folders:

- Point 3—the RTCESL library source folder contains the header files for the mathematical and control functions used in the project. The theory of using and applying these functions is described in the user's guides specific for each library. Download the user's guides from www.nxp.com/rtcesl.
- Point 4—the *sdk* folder contains the startup routines, the system initialization and clock definition, the linker file, and the header file for the MCU. It also contains the basic CMSIS routines for the interrupt handling.
- Point 5—the *src* folder contains the application source code.
- Point 6—the output file generated by the compiler, ready to use with the default debugger (P&E Micro—OpenSDA). This debugger provides a virtual serial port on the host computer. Plug the cable's USB mini-B connector to the J2 connector on the controller card and the USB type A connector to the computer before debugging. The debugger can be changed to another one in the project options by right-clicking Point 1, selecting "Options" and clicking "Debugger". Start the project debugging by clicking Point 7 (Figure 19).

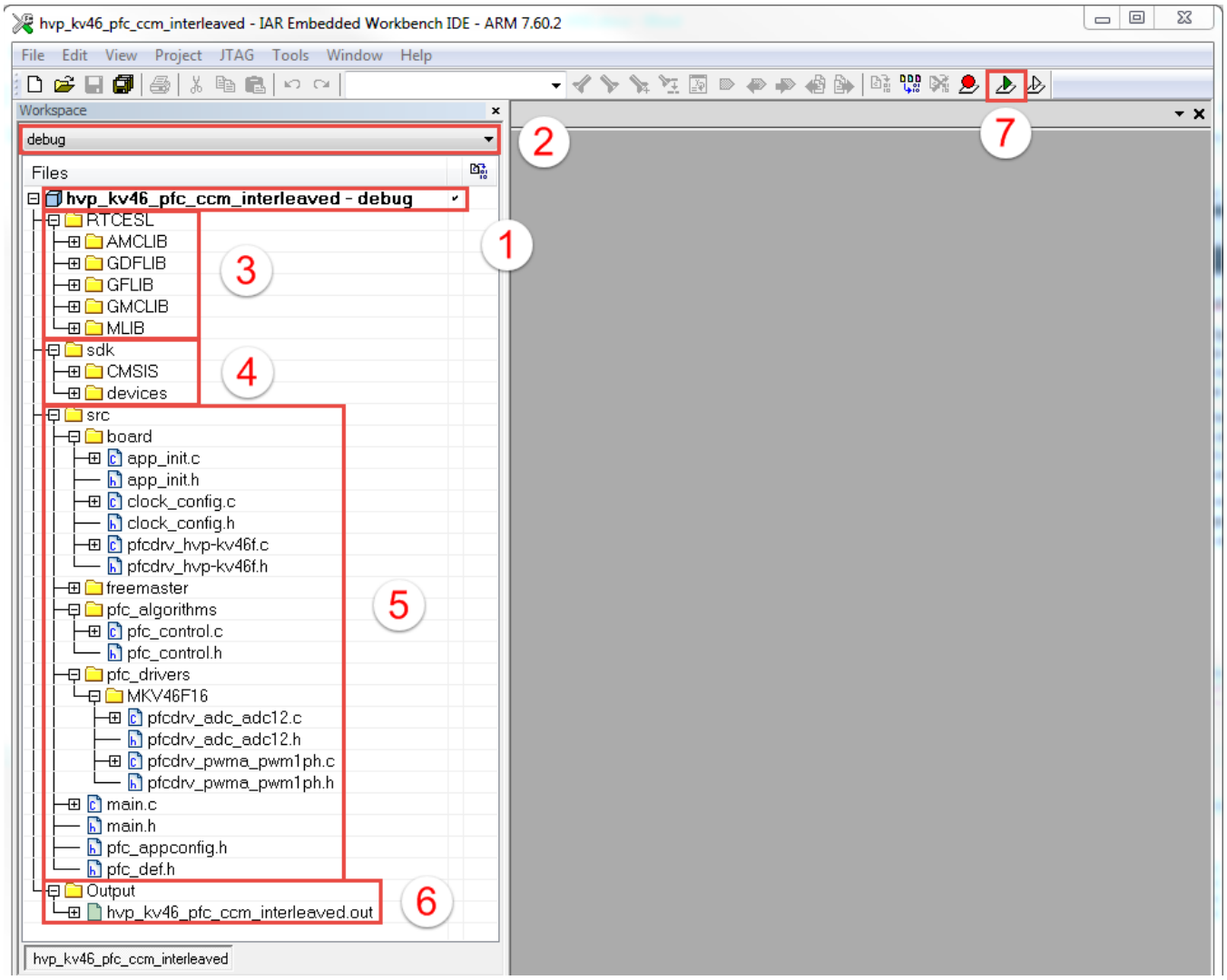

Figure 19. Hardware setup

5.2.1. Application upload without IDE

To load the generated application directly to the target MCU without the IAR Embedded Workbench IDE installation, perform these steps (applicable for Windows® OS):

1. Open Windows OS Explorer.
2. Locate the generated S-record file *hvp_kv46_pfc_ccm_interleaved.srec*.
3. Drag and drop or copy and paste the selected S-record file to the MSD removable drive with the volume labelled as the target hardware (HVP-KV46F15).
4. After a successful programming, the embedded application executes automatically.
5. Reconnect the target device.

5.3. Application control

The application can be controlled remotely from a computer using FreeMASTER. The computer is connected to the controller through a standard A-B USB cable. The USB connector on the main board (J4) is galvanically isolated from the high voltage and no additional isolation is required.

Open the FreeMASTER project file *pfc_interleaved_ccm_kv4x.pmp* located in the *build/iar* project folder. After the project opens, the control page shows up, as shown in this figure:

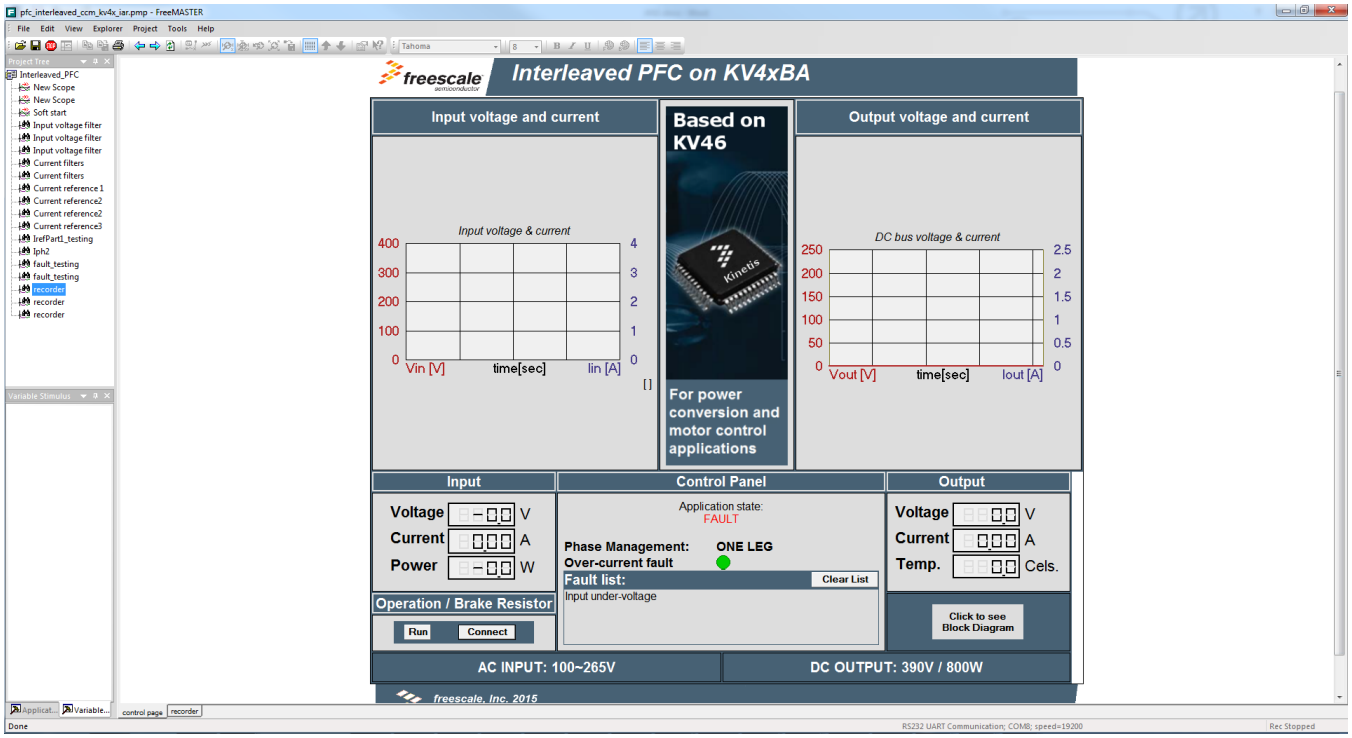

Figure 20. FreeMASTER control page

A proper communication port must be set in the “Project, Options, Comm” menu. The assigned virtual COM port can be checked in the “Devices and Printers” menu in Windows OS. The driver for the virtual COM port must be installed before connecting the USB cable to the controller card for the first time. The controller card contains the CP2102 USB-to-COM port converter from Silicon Labs®. The driver can be downloaded from www.silabs.com.

The communication with FreeMASTER can be started from the “File, Start Communication” menu, by pressing CTRL + K, or pressing the “Stop” button in the main menu. If the host PC is successfully connected to the controller card, the COM port and the communication speed is displayed in the bottom right corner of FreeMASTER. Everything is now prepared for the application demonstration.

The PFC operation is started by pressing the “Run” button on the control page. The load is connected after pressing the “Connect” button. The key application parameters are displayed on the control page and you may easily verify the performance of the PFC algorithm. The PFC operation is stopped by pressing the “Stop” button on the control page.

6. PFC integration

The stand-alone reference software is developed using the High Voltage Motor Control Platform (HVP-MC3PH). However, the integration of the PFC algorithm into any other applications (motor control, switch mode, power supply) is straightforward.

The access to the PFC application is done using the global PFC operation control structure *gsPFCDrive*. This structure contains the main PFC control structure *sInterleavedPFC*, the PFC application control structure *ui16PfcControl*, the PFC application status structure *ui16PfcStatus*, and the fault identification structure *sFaultId*. All PFC application control structures are described in detail in the following section.

6.1. PFC application control structures description

The *gsPFCDrive* structure contains the variables and constants used for the PFC application control, such as the control structure of the voltage controller, the current controllers, the control structure for the average filters of the input voltage, two boost currents, and the DC-Bus voltage. It also contains the variables necessary for the PFC application control, such as the duty cycles, the measured quantities, its filtered values, and other. The parameters of the PI controllers and the average filters are calculated according to the *Average Current Mode Interleaved PFC Control* (document AN5257) and set in the *pfc_appconfig.h* file.

```
typedef struct
  {
       bool_t                          bRunStop;
       bool_t                          bBrakeRes;
  } PFC_CONTROL_T;
```

The code above defines the control bits for the PFC control from the application state machine. The meaning of each bit is as follows:

- *bRunStop*—PFC application control bit, 0—stop the PFC operation, 1—start the PFC operation.
- *bBrakeRes*—brake resistor control bit, 0—resistor disconnected, 1—resistor connected.

```
typedef struct
{
       bool_t        bRunStop;
       bool_t        bSoftStart;
       bool_t        bFault;
} PFC_STATUS_T;
```

The code above defines the status bits to inform the application state machine about the PFC application status. The meaning of each bit is as follows:

- *bRunStop*—PFC application status bit, 0—stop state is in progress, 1—run state is in progress.
- *bSoftStart*—soft start status bit, 0—soft start state not active, 1—soft start state is in progress.
- *bFault*—fault status flag, 0—PFC application is not in fault state, 1—PFC application is in the fault state.

```
typedef uint16_t PFCDEF_FAULT_T;
```

The code above defines the status bits to identify the fault that causes the fault state. The meaning of each bit is defined using these constants:

```
#define PFC_FAULT_I_PFC_OVER      0   /* OverCurrent fault flag */
#define PFC_FAULT_U_IN_UNDER      1   /* Input undervoltage fault flag */
#define PFC_FAULT_U_IN_OVER       2   /* Input overvoltage fault flag */
#define PFC_FAULT_U_DCBUS_UNDER   3   /* dc-bus undervoltage fault flag */
#define PFC_FAULT_U_DCBUS_OVER    4   /* dc-bus overvoltage fault flag */
#define PFC_SS_FAULT              5   /* Soft start fault flag */
```

7. Efficiency measurement

The Total Harmonic Distortion (THD) was measured at 115 VAC and 230 VAC. The measurement is summarized in the following tables. An electronic load in the constant resistance mode was used for the measurements. The power limits vary with the input voltage. The power limits of the power stage are shown in Figure 4 in the *Freescale High-Voltage Motor Control Platform User's Guide* (document HVPMC3PHUG).

Table 2. Measurement at 115 V/60 Hz

Output power [W]	Efficiency [%]	Power factor [-]	THD [%]
100	91.5	0.97	19.5
200	93.4	0.98	9.9
400	93.5	0.99	4.5
600	93.4	1	3.5
750	92.5	1	2.7

Table 3. Measurement at 230 V/50 Hz

Output power [W]	Efficiency [%]	Power factor [-]	THD [%]
100	92.3	0.93	33
200	95.2	0.95	29.6
400	96.4	0.97	18.4
600	96.7	0.98	7
800	96.4	0.99	5.5

8. Conclusion

This application note describes the implementation of the average current control mode PFC on the KV46xx MCU. The ARM-based KV-series MCUs are targeted for the power management and motor control applications due to the dedicated peripherals. Even if the application code demonstrates a stand-alone PFC operation, the code is written with an intention to be easily integrated into other applications. The design of the control loops is explained with a full theory and a practical example. The application source code is provided together with the application note.

9. References

- *Average Current Mode Interleaved PFC Control* (document AN5257)
- *Freescale High-Voltage Motor Control Platform User's Guide* (document HVPMC3PHUG)
- *HVP-KV46F150M User's Guide* (document HVPKV46F150MUG)

The Real-Time Embedded Software Libraries are located at www.nxp.com/rtcesl. See these documents for a specific reference when using this application note:

- *Set of General Digital Filters for Cortex M4 Core User's Guide* (document CM4GDFLIBUG)
- *Set of Math Functions for Cortex M4 Core User's Guide* (document CM4MLIBUG)
- *Set of General Functions for Cortex M4 Core User's Guide* (document CM4GFLIBUG)

For a current list of all documentation, visit www.nxp.com.

10. Revision history

The following table summarizes the changes done to this document since the initial release.

Table 4. Revision history

Revision number	Date	Substantive changes
0	11/2016	Initial release.

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Document Number: AN5355
Rev. 0
11/2016