Firmware for STM32F334xx microcontroller on STEVAL-DPSLLCK1 LLC full bridge converter with SR

Introduction

The STSW-DPSLLCK1 application firmware package for the STEVAL-DPSLLCK1 3 kW converter reference design helps developers evaluate and customize digital power design of LLC resonant converters and assess the performance of the STM32F334xx microcontroller in combination with featured power and analog components.

The firmware example implements the control algorithm and required protection functions for constant voltage mode operation. The STM32F334xx microcontroller with high resolution timer drives primary side power MOSFETs with 50% duty cycle PWM signals. An appropriate dead time for each leg is included to ensure zero voltage switching and avoid input voltage shoot-through. The PID voltage regulator running at 50 kHz provides the PWM switching period of primary side devices to change the voltage gain of the resonant tank and regulate the output voltage to the desired value.

The resonant tank is designed to always operate in the inductive region and then in soft switching for the entire frequency range. When the board is powered on, a switching period ramp-up at decreased frequency is performed to avoid current spikes. An adaptive synchronous rectification (A-SR) algorithm based on V_{DS} sensing is used to drive the secondary side power MOSFETs and reduce conduction losses. Burst mode operation is adopted at light loads to reduce losses due to circulating current and increase converter efficiency.

The control firmware also provides fast overcurrent protection, input and output under- and overvoltage protection, and overtemperature protection. An additional PWM signal is used to drive a cooling fan at a speed that depends on output load and heat-sink temperature.

Finally, a basic user UART interface allows control of the main operating parameters of the converter and enable or disable control features such as open loop operation.

IAR embedded workbench for ARM is used to write, compile and debug the source code, but other toolchains can be used.

RELATED LINKS

AN2450: LLC resonant half-bridge converter design guideline

AN2644: An introduction to LLC resonant half-bridge converter

AN4720: Half bridge resonant LLC converters and primary side MOSFET selection

UM2348: Getting started with the STEVAL-DPSLLCK1 evaluation kit for the 3 kW full bridge LLC digital power supply

1 Overview

1.1 Package content

The DSMPS 3kW - LLC STM32F334x firmware package contains the following folders:

- Documentation - with compressed html and latex help files created by Doxygen
- Drivers - with HAL, LL, CMSIS and BSP drives
- FullBridge_LLC_Project - which contains a readme file, source files, linker configuration files (.icf), microcontroller startup file (.s) and IAR Embedded Workbench related files, such as the workspace file (.eww), project file (.ewp) and debugger setting file (.ewd)

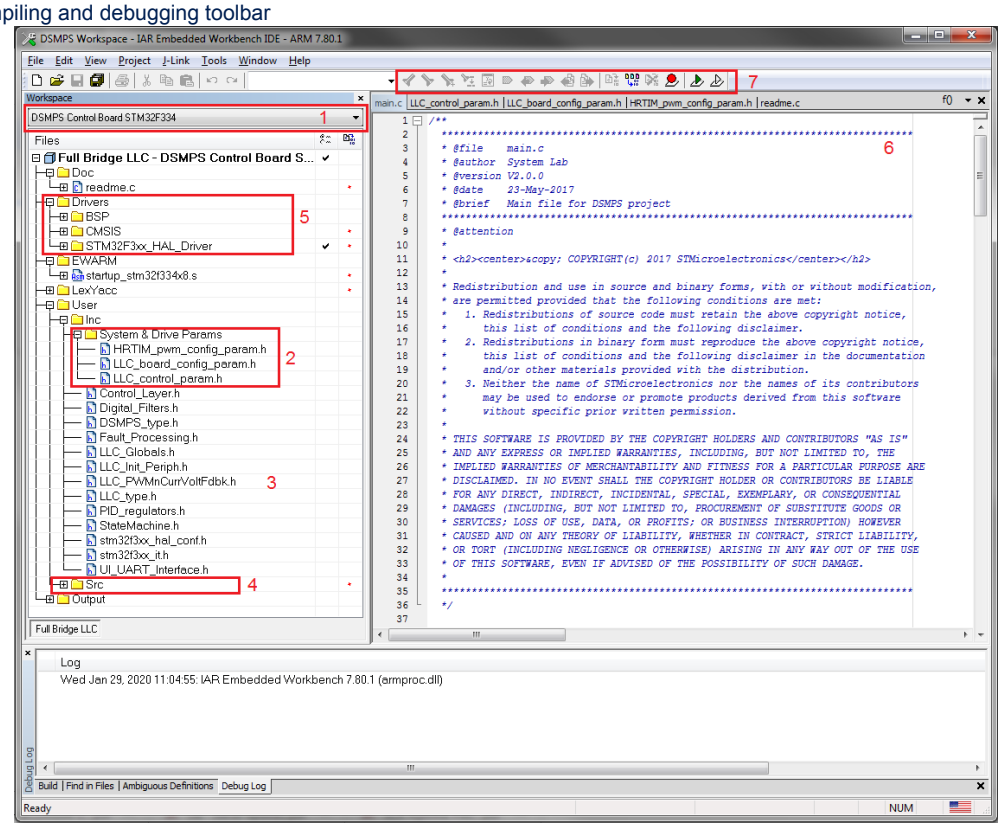
Compiled files will be placed inside a folder named as the configuration used in the workspace (DSMPS Control Board STM32F334 by default).

1.2 Running FB LLC FW example

To run the DSMPS 3kW - LLC STM32F334x FW example, the workspace file (DSMPS Workspace.eww) should be opened with IAR Embedded Workbench v7.80 or higher.

Figure 1. IAR Embedded Workbench window

1. Configuration setting
2. Main configuration files
3. Other included header files
4. Source files
5. Library files
6. Text editor
7. Compiling and debugging toolbar



The left side of the image shows the workspace and related files. The System & Drive Params folder contains the main user configuration files to enable or disable certain features.

The default configuration setting is DSMPS Control Board STM32F334, which is specific for the STEVAL-DPS334C1 control board used to run the power application. Other configurations for NUCLEO-F334R8 and 32F3348DISCOVERY boards are only useful for debug purposes as the pin-out and some MCU resources are different.

The firmware can be compiled and flashed using the indicated toolbar. To reprogram the board, you need to configure your preferred debugger tool (e.g., ST-LINK) in the IAR Embedded Workbench option menu by setting the SWD interface and connecting the adapter board.

RELATED LINKS

See UM2348: Getting started with the STEVAL-DPSLLCK1 evaluation kit for the 3 kW full bridge LLC digital power supply

1.3 STM32Cube firmware

The STSW-DPSLLCK1 example uses STM32Cube HAL (Hardware Abstraction Layer) libraries to configure MCU peripherals. In some parts of the code, STM32Cube LL (Low Level) libraries are used for performance purposes. These LL drivers are built on the standard CMSIS library.

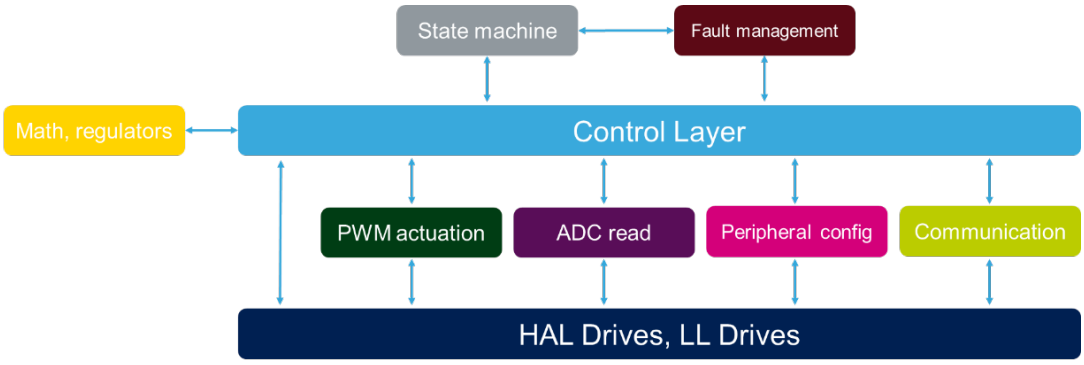
1.4 FB LLC firmware architecture

The FB LLC firmware example includes all the software functions to configure microcontroller peripherals and execute the control algorithm. These functions are divided into different files and can be considered as part of a sub-block.

Basic sub-blocks use HAL and LL drivers in function definitions to modify MCU registers (e.g., PWM actuation, peripheral configuration, etc.). These functions can be easily rewritten or modified by the user for firmware customization. The control layer calls functions of basic sub-blocks to perform the control algorithm.

The state machine and fault management blocks are on top of the control layer.

Figure 2. FB LLC firmware architecture



1.5 UI communication

The FB LLC firmware example can be executed without any communication interface, and the power conversion begins as soon as an input voltage within the defined range is detected.

A simple UART UI interface is available to enable some real-time functionality such as changing control parameters and detecting and clearing faults.

2 STEVAL DPSLLCK1 firmware application

2.1 File description

This section lists the most important source files of the FB LLC firmware application example. The same information is available in the readme file.

LLC_control_prm.h: this file contains the most important definitions the user can change to enable/disable some features and protections, as well as to modify PID default control parameters, task execution frequencies, thresholds used for SR and burst mode, switching frequency range (expressed in Hz), etc. Most of the possible customizations can be made by modifying this file.

LLC_board_config_param.h: this file contains definitions of input/output ports, MCU resources and conversion factors of external sensors (Hall sensor, voltage dividers, temperature, etc.). If the solution is customized, this file should be modified accordingly.

HRTIM_pwm_config_param.h: this file contains definitions used for HRTIM configuration such as clock frequency, pre-scaler value, maximum, minimum and default dead time value (expressed in ns), polarity of each HRTIM output (according to the driver used in power board), default burst mode duration, etc. All these parameters can be modified by the user. The file also contains the definitions of some macros used to convert time delays in an HRTIM tick number, and they should not be modified.

DSMPS_type.h: contains type definition of variables used for a generic DSMPS control and not strictly related to LLC converter. These types include PID structure, State machine, LED configuration structure, driver pre-charge structure and fault error codes, as well as period and blinking number for LEDs for generic PFC and DC-DC topologies.

LLC_type.h: this file extends type definitions of DSMPS_type.h file with specific types used for the LLC converter. The DCDC_MeasureStruct_t type defines the structure of ADC acquisitions: the variable of this type (named DCDC_MeasureStruct) is filled with the last ADC measurements for both regular and injected acquisitions. Other important definitions are the enumeration type of SR MOSFET leg (SR_LEG1 and SR_LEG2) and DCDC_ConfigParamStruct_t type which contains all parameters that can be modified on-the-fly by the user either via debugger or user interface.

Control_Layer.c and **Control_Layer.h**: contain definitions and prototypes of functions used at high level to enable/disable PWM outputs, enable/disable burst mode, enable/disable SR, execute voltage control loop, set PWM frequency in open loop mode, initialize control parameters, set or check a time delay, etc.

LLC_Globals.c and **LLC_Globals.h**: contain definitions and declarations of the main variables used for control algorithm and peripheral configuration (Handles). The main variables are: hPWMPeriod (PWM period in HRTIM tick number), hVout_Reference (output voltage reference in 0 - 4095 range), PID_Vout_InitStructure (PID regulator structure with all coefficients, previous error and accumulator), DCDC_MeasureStruct (which contains last sensed measurements from ADC channels), hPWMDeadTime (value of LLC PWM dead time in HRTIM DT ticks, it is used because dead time can be changed also on-the-fly), SR delays for fixed SR driving, counters used for time scheduling (wait time, start-up time, etc.), boolean variables used for protection enabling and DCDC_ConfigParamStruct variable (of DCDC_ConfigParamStruct_t type), which stores all configuration parameters that can be changed on-the-fly by UART interface.

PID_regulators.c and **PID_regulators.h**: contain definitions and prototypes of PID regulators (both standard and extended controllers - refer to Section 2.5 PID regulator) and related functions. There are generic PID controllers with anti-windup and saturation features, and functions that can be used to change the proper PID field, such as coefficients, upper limit, lower limit, etc. The DCDC_PID_Init() function and its extended version are used to initialize PID structure with default values defined in LLC_control_param.h file.

LLC_Init_Periph.c and **LLC_Init_Periph.h**: contain definitions and prototypes of functions used to configure and initialize MCU peripherals like ADC, HRTIM, COMPs, DAC, etc. These functions may be modified for user purposes.

Digital_Filters.c and **Digital_Filters.h**: contain definitions and prototypes of functions used to perform digital filters. For LLC converter, only a simple low pass filter is used, which implements the formula: yk = y(k-1) - (y(k-1)>>n) + (xk>>n), where yk is the filter output at discrete time k, xk is the input at the same time, y(k-1) is the previous output and n is a shift value.

StateMachine.c and **StateMachine.h**: contain definitions and prototypes of functions used to perform the state machine and its related functions. STM_StateMachineTask() function executes the state machine, other functions are used to modify the status of the state machine (DSMPS_State variable) that is accessible only by get/set methods. Other private variables are bConverterEnabled, which identifies if the power conversion can start (its value depends on the default setting or communication commands) and bStartUpComplete flag, which is automatically set when the start-up procedure is completed correctly. In this file, functions related to LED driving (User LED or Fault LED) are also defined.

LLC_PWMnCurrVoltFdbk.c and **LLC_PWMnCurrVoltFdbk.h**: contain definitions and prototypes of functions used to change PWM switching frequency, to drive SR stage, update burst mode, change dead time, etc. Most of these functions are located in the CCM RAM memory, using #pragma instruction, to speed up their execution as they are called via a high frequency task to update HRTIM registers after control algorithm execution. Functions used to drive the fan with an auxiliary PWM and to get SR MOSFET Vds injected measurements are also present.

Fault_Processing.c and **Fault_Processing.h**: contain definitions and prototypes of functions used to check protections as well as HRTIM fault interrupt routine used for fast overcurrent protection and AWD interrupt for output overvoltage, if AWD is enabled.

UI_UART_Interface.c and **UI_UART_Interface.h**: contain definitions and prototypes of functions used for user UART communication. These functions include UART configuration, hardware initialization, UART receiving DMA interrupt and complete transfer callback. The scanning (the input stream conversion from characters to meaningful tokens) and the parsing (the interpretation of this tokens and the definition of related rules) are made thanks to lex and yacc files (lexter.l and parser.y) that, once properly compiled, generate lex.yy.c, y.tab.c and y.tab.h files (put inside LexYacc folder), which are recognized by the C program and used for string processing.

stm32f3xx_hal_conf.h: the HAL configuration file. The use of a new HAL module is enabled by modifying this file.

stm32f3xx_it.c and **stm32f3xx_it.h**: contain definitions and prototypes of interrupt service routines. The interrupts configured relate to the scheduling of the control algorithm, the scheduling of low frequency tasks, SysTick and UI communication.

stm32f3xx_hal_msp.c: contains function definitions to configure hardware resources. These functions are called by HAL library functions used to configure MCU peripherals.

main.c: contains main() function that represents the starting point of the program. Inside the main(), after HAL initialization, all functions needed to configure the MCU (defined in Init_Periph.c file) are called, according to the functionalities enabled in LLC_control_prm.h file. After peripheral configuration, CTR_InitEnvironment(), defined in Control_Layer.c, initializes main control variables, such as PID structure, configuration structure, filtered values, etc. Inside the while(1) loop, FLT_FaultCheck(), STM_StateMachineTask() and LED_Task() functions are called. Interrupt service routines will interrupt this infinite loop when the corresponding event occurs.

2.2 Interrupt and tasks

Most control tasks are performed inside specific interrupt service routines, called with a fixed time scheduling or after an external event, and they have different priorities according to their function. The main control loop, synchronous rectification algorithm and frequency start-up are performed inside the TIM6 update interrupt service routine, set by default at 50 kHz. A lower frequency task is scheduled by the TIM16 update event, set at 100 Hz. Inside this interrupt routine, the calculation of low pass digital filter for input voltage and temperature measurements is performed, as well as the regulation of fan speed and the enabling/disabling of SR signals according to the load applied.

All interrupt routines are placed in stm32f3xx_it.c file, except the HRTIM fault interrupt, which is placed in Fault_processing.c file.

Task execution frequencies can be changed in LLC_control_param.c file:

```
/**** Voltage Control loop frequency ****/

#define VOLTAGE_CONTROL_LOOP_FREQ_HZ((uint32_t)50000)
/**< execution frequency of Vout control loop in Hz */


#define VOLTAGE_CONTROL_TIM_PRS ((uint16_t)0)
 /**< prescaler of TIM that schedules Vout control loop */
```

```
/**** Temperature acquisition and fan control frequency ****/
```

```
#define LOW_FREQUENCY_TASK_FREQ_HZ ((uint16_t)100)
/**< execution of low frequency task in Hz: fan speed, SR turn-on/off, Vin and Temp
Filtering, update config params */
```

```
#define LOW_FREQUENCY_TASK_TIM_PRSC ((uint16_t)9999)
/**< prescaler of TIM that schedules execution of low frequency task in Hz: fan, SR enable/
disable, Vin, Temp Filtering and update config params */
```

Asynchronous tasks can be performed after an overcurrent or overvoltage detection (the last one if AWD is enabled; for further details, refer to Section 2.8 Faults and protections).

Also UI communication related functions are performed asynchronously when a new command is received.

Finally, the timings of some slow procedures (start-up, wait time, output undervoltage validation time and LED blinking) are scheduled using some counters decreased inside the Systick interrupt routine (set at 1 ms).

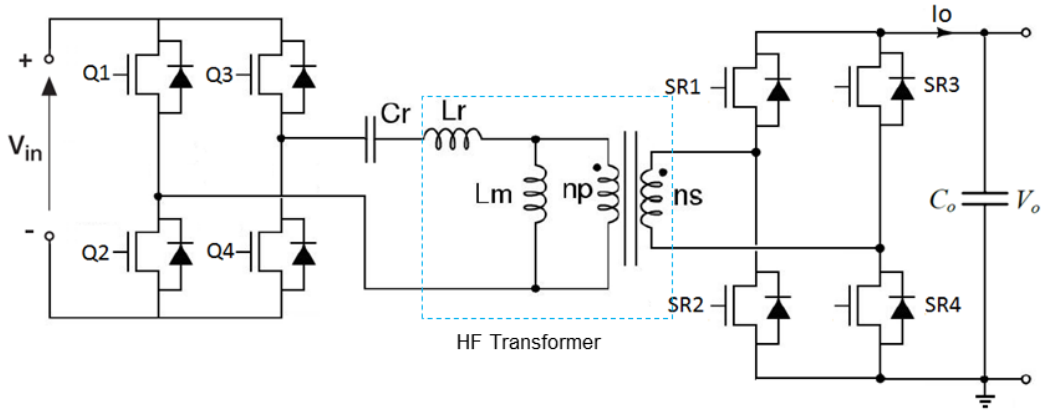
Table 1. Firmware main tasks

Task name	Priority	Execution frequency	Routine	Description
Main control loop	High	50 kHz	TIM6 interrupt routine	Voltage mode: PID regulator to calculate the switching period to control output voltage
Frequency decrease	High	50 kHz	TIM6 interrupt routine	Start-up procedure to decrease switching frequency linearly up to the reference value to avoid overcurrent. It is performed at start-up phase instead of the main control loop.
Synchronous Rectification and light load burst mode	High	50 kHz	TIM6 interrupt routine	Adapts turn-off PWM edges of SR MOSFETs to the internal diode conduction time. Enables/disables light load burst mode depending on the output current.
Fan speed regulation and SR enable/ disable	Medium	100 Hz	TIM16 interrupt routine	SR enabling/disabling is made according to the load current, as well as the fan speed regulation.
Slow protection checks	Low	-	Main loop	Compare measured values with high and low thresholds and manage state machine accordingly. Fault checked: • Input under/over voltage • Output under/over voltage • Overtemperature • Output overcurrent
Fast protection checks	Very high (HW)	Immediately after COMP delay and digital filter	HRTIM Fault interrupt routine	Fast hardware protection and relative Irq handler. Fault checked: • Resonant current overcurrent
Serial communication	Very low	Asynchronous	DMA interrupt on data received	Manages UART interface to change the main control parameters.

2.3 Control loop

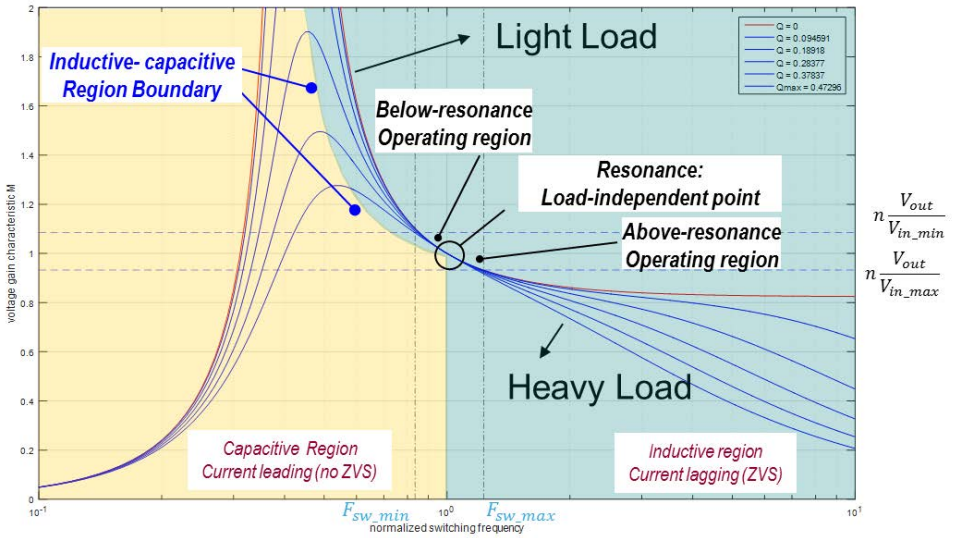
The STSW-DPSLLCK1 firmware example configures the STM32F334 microcontroller on the control board of the STEVAL-DPSLLCK1 kit to drive both primary and secondary side MOSFETs to control the output voltage (Vo), starting from a constant input voltage (Vin).

Figure 3. STEVAL-DPSLLCK1 topology



For this purpose, primary side high voltage power MOSFETs (Q1 to Q4) are driven two-by-two (Q1-Q4 and Q2-Q3) with 50% PWM duty cycle and an appropriate dead time. In this way, a square voltage waveform between –Vin and Vin is applied to the resonant tank (composed of Cr, Lr and Lm). According to resonant tank design, modifying the switching frequency can change the voltage gain and regulate the output voltage, making the converter always work in the inductive region and ensure ZVS operation, as shown below.

Figure 4. LLC voltage gain characteristic



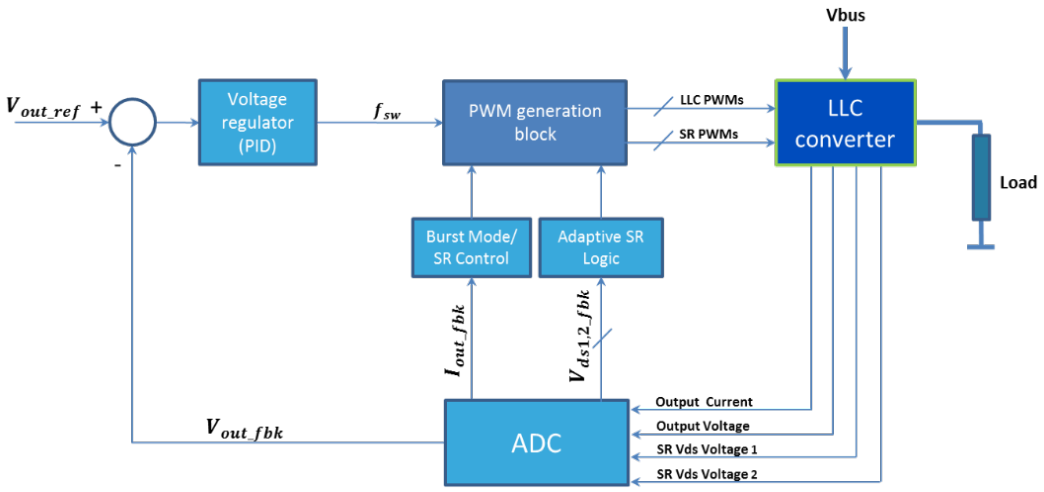
Switching frequency is changed according to the feedback control loop, in which the Vout measurement is fed back and compared to the output voltage reference; the voltage error obtained is processed by a standard PID regulator, whose output represents the new switching period for PWM generation block (HRTIM). Control loop algorithm is executed at fixed frequency (50 kHz by default) and ensures a constant output voltage over the entire load range. The user could also modify the firmware to regulate the output current instead of the output voltage (i.e., for battery charger applications) feeding back the output current measurement (sensed by the isolated Hall sensors present in the board), keeping in mind that the PID parameters have to be changed accordingly.

The STM32F334 microcontroller also generates PWM driving signals for secondary side switches performing Synchronous Rectification to reduce conduction losses. Also in this case, power MOSFETs are driven two-by-two (SR1-SR4 and SR2-SR3) to perform the rectification of the transformer output voltage.

The Adaptive SR technique adapts driving signal duration to the conduction time of the power MOSFET body diode thanks to Vds voltage sensing of low side switches. At low load conditions, SR does not provide any benefits (switching losses are comparable or higher than reduction of conduction losses), so it is disabled if output current is lower than 5 A.

An LLC converter requires a small current needed to magnetize the transformer, even if no load is applied. For this reason, at very low load condition, burst mode is activated using a special feature of HRTIM; in this way some PWM pulses, inside a defined burst period, are skipped to reduce conduction losses.

Figure 5. Control loop block diagram



RELATED LINKS

AN2450: LLC resonant half-bridge converter design guideline

AN2644: An introduction to LLC resonant half-bridge converter

2.4 State machine

The STSW-DPSLLCK1 firmware package includes a state machine that manages the converter operation control. The tasks executed in the code and the functions that can be called depend on the state machine current state, which changes according to the previous state and if some conditions are verified.

Figure 6. **LLC state machine**

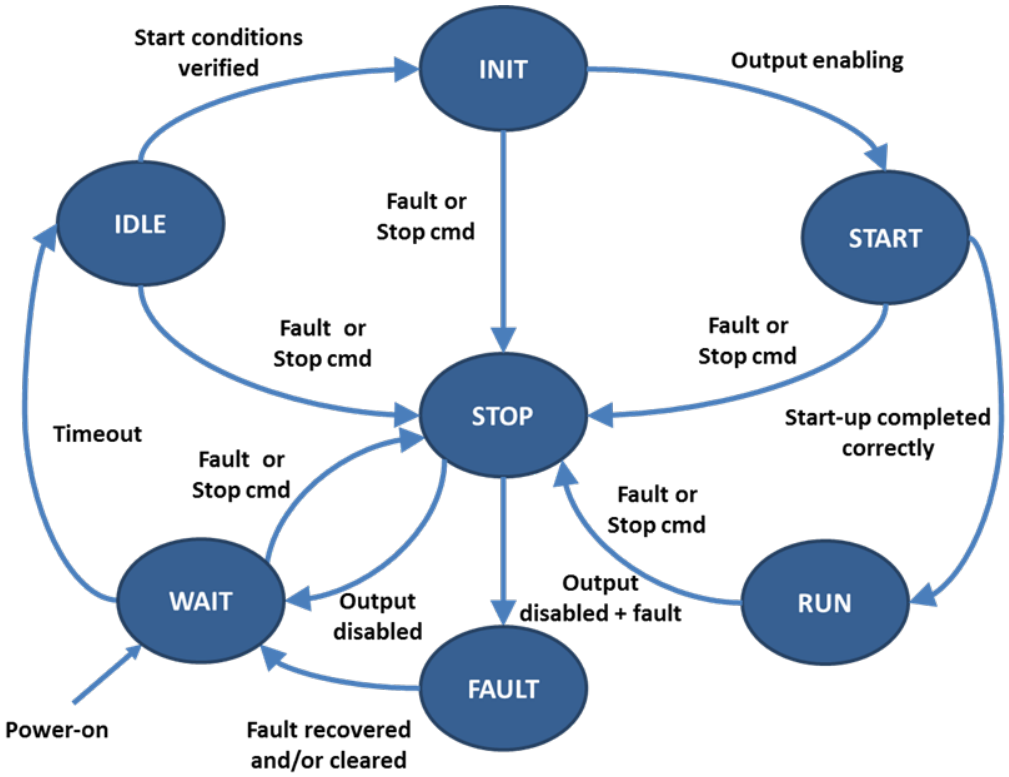


Table 2. **State machine description**

State	Description
WAIT	Persistent state where the application is intended to stay, for a defined wait time, after fault conditions disappear or at power-on. The following state is normally IDLE.
IDLE	Persistent state; the following state can be INIT if all the required conditions (input voltage and/or external command) are satisfied or STOP if there is a fault condition.
INIT	Pass-through state: the code is executed only once between IDLE and START states to initialize control variables and enable outputs. The following state is normally START but it can also be STOP if there is a fault condition. The system can remain in this state if the gate driver boot capacitor pre-charge, needed to supply HS power MOSFET by gate driver, is enabled.
START	Persistent state where the converter start-up is intended to be executed. The following state is normally RUN as soon as start-up procedure is completed correctly. The following state is STOP if there is a fault condition.
RUN	Persistent state where the converter is running normally. The following state is STOP if there is a fault condition or a stop command is received.
STOP	Pass-through state where PWMs are disabled. The following state is FAULT if a fault stops the converter, WAIT otherwise (external command received).
FAULT	Persistent state after a fault. The following state is normally WAIT until faults are cleared or recovered (not all faults can be recovered without a system reset).

The control board status LED (blue) indicates the state machine state:

- OFF in all the states in which PWMs are disabled
- IDLE when the LED blinks slowly and indefinitely
- INIT and START when the LED blinks quickly
- ON in RUN state

2.5 PID regulator

The Proportional, Integral and Derivative (PID) regulator is used to eliminate the error between output voltage and its reference.

The PID regulator output defines the counter overflow value to be set in the HRTIM peripheral (PER register), according to the clock frequency (and then resolution) selected, and it represents the new switching period after the next update event.

In time domain, the PID regulator output is given by the following equation:

$$r(t) = K_p \times \epsilon(t) + K_i \times \int_0^t \epsilon(t)\,dt + K_d \times \frac{d}{dt}\epsilon(t) \quad (1)$$

Where:

- r(t) is the regulator output at time t
- ε(t) is error of the system at time t
- K_p is the proportional coefficient
- K_i is the integral coefficient
- K_d is the derivative coefficient

For a digital control loop performed at a fixed frequency (i.e., in discrete time domain), the equation becomes:

$$r(t_k) = K_p \times \epsilon(t_k) + K_i \times \sum_{j=0}^{k} \epsilon(tj) \cdot T_s + K_d \times \frac{\epsilon(t_k) - \epsilon(t_k - 1)}{T_s} \quad (2)$$

Where:

- $r(t_k)$ is the regulator output at discrete time t_k
- $\varepsilon(t_k)$ is the error of the system at discrete time t_k
- T_s is the sampling time (control period)

To reduce CPU load, the sampling time is directly part of the integral coefficient, thus avoiding an extra multiplication and division in the previous equation.

As an accumulative term, the integral part is used in the algorithm: increasing the loop time decreases its effect (accumulation is slower and the integral action on the output is delayed). Inversely, decreasing the loop time increases its effect (accumulation is faster and the integral action on the output is increased). Therefore the sampling time (i.e., control frequency) has to be adjusted before changing PID coefficients.

In theory, the higher the sampling rate, the better the regulation. In practice, the following consideration must be made:

- CPU load increases accordingly
- There is no need for a sampling time lower than the refresh rate of external sensors
- PID is a linear regulator, thus for a resonant converter like LLC in which switching frequency is the control variable, performing the algorithm at variable frequency (e.g., switching frequency or its sub-multiple) could generate unexpected results, unless updating T_s, and then K_i and K_d at each control loop, needlessly increasing the CPU load
- Resonant converters can switch at very high frequency (several hundreds of kHz), whereas controlled variables (output DC voltage and DC current) have a relatively slower dynamic, so there is no need to control at switching frequency and the CPU load would in any case be unacceptable

For the above reasons, a 50 kHz control frequency has been chosen for this application for good regulation performance.

The PID regulator function is implemented in PID_regulators.c file and it has the same structure as the regulator used in STM32 motor control SDK. The output of this regulator is a signed 16-bit variable (int16_t), so it is in range: [-(2^{15}-1); 2^{15}-1], as one of the bits is used to indicate the sign. In the LLC application, the output of the regulator represents the new overflow value for HRTIM counter, which is an unsigned 16-bit variable (uint16_t) in range [0; 2^{16}-1]. For this reason, according to the pre-scaler set in HRTIM clock configuration and minimum switching frequency chosen, a signed 16-bit variable might be not enough. For example, if the maximum resolution (HRTIM clock frequency set at 4.608e9 Hz) is adopted, the minimum switching frequency given when HRTIM_PER is set at its maximum value (0xFFDF = 65503, according to the reference manual) which is around 70.3 kHz, but the standard regulator, whose output can reach (2^{15}-1)= 32767 maximum, cannot guarantee this minimum frequency. For this reason and to avoid losing resolution (setting a pre-scaler greater than necessary), an extended version of PID regulator with a 32-bit output is provided in the same source file together with all the related functions.

To enable/disable the extended PID calculation, keeping in mind the maximum output required according to minimum switching frequency and HRTIM clock frequency, the relevant definition in LLC_control_param.h file is uncommented (default) or commented accordingly:

```
#define USE_EXTENDED_PID
```

It is also possible to disable differential term calculation, for both standard and extended PID regulators, commenting the following lines in the PID_regulators.h file:

```
#define DIFFERENTIAL_TERM_ENABLED
#define DIFFERENTIAL_TERM_ENABLED_EX
```

The definitions of regulator coefficients and their divider (integer calculation has been adopted, divisors should be a power of two to speed-up code execution), are placed in LLC_control_param.h file.

```
/*--------------------------- PID CONTROL PARAMETERS --------------------------*/
/************** VOUT PID-CONTROLLER INIT VALUES **************/

#define PID_VOUT_KP_DEFAULT ((int16_t)3000)
/**< default Kp value of PI regulator */

#define PID_VOUT_KI_DEFAULT ((int16_t)1000)
/**< default Ki value of PI regulator */

#define PID_VOUT_KD_DEFAULT ((int16_t)1000)
/**< default Kd value of PI regulator - if used */

#define PID_VOUT_INTEGRAL_UPPER_LIMIT (PID_VOUT_UPPER_LIMIT * VOUT_KIDIV2)
/**<upper limit of integral term of PI regulator */

#define PID_VOUT_INTEGRAL_LOWER_LIMIT  (PID_VOUT_LOWER_LIMIT * VOUT_KIDIV2)
/**<lower limit of integral term of PI regulator */

#define PID_VOUT_UPPER_LIMIT(HRTIM_MAX_PWM_PERIOD)
/**< output upper limit of PI regulator */

#define PID_VOUT_LOWER_LIMIT(HRTIM_MIN_PWM_PERIOD)
/**< output lower limit of PI regulator */

#define VOUT_KPDIV2 ((uint16_t)(256))
/**< default Kp divider value of PI regulator */
```

```
#define VOUT_KIDIV2 ((uint16_t)(4096))
/**< default Ki divider value of PI regulator */

// Vout PID parameter dividers

#define VOUT_KDDIV2 ((uint16_t)(2048))
/**< default Kd divider value of PI regulator - if used */
```

Regulator parameters can also be changed via UART interface or acting directly on DCDC_ConfigParamStruct variable via debugger (and setting the bConfigurationChanged field to TRUE to update the changes).

2.6 Start-up procedure

One of the most critical processes in the control of LLC converter is the start-up procedure. At converter turn-on, a large current spike due to the discharged magnetizing inductance of the transformer can occur if the procedure is not well controlled. A common practice consists in forcing a low voltage gain at the beginning, that is to turn on devices with a high switching frequency, usually 2 or 2.5 times the resonant frequency, while also ensuring the operation remains in the inductive region.

Initially, the control loop is disabled and the switching frequency is set high (380 kHz) and decreased over 500 ms to the minimum value (120 kHz). When the output voltage reaches the desired value of 46 V, the control loop is closed and the switches are driven with the frequency given by PID regulator. To avoid discontinuity when the loop is closed, the integral term of PID regulator is set equal to the corresponding value of last PWM frequency. When the regulation is ongoing, switching frequency can change over a smaller operating range (120 to 250 kHz).

The output threshold at which the control loop is closed is defined in the StateMachine.c file:

```
#define VOUT_MIN_CLOSED_LOOP OUT_VOLT_ADC_VALUE(46)
/**< minimum voltage in [V] to close the control loop during the start-up phase */
```

Maximum, minimum and start-up frequencies are defined in LLC_control_param.h file:

```
#define HRTIM_MAX_PWM_FREQ_START_UP_HZ 380000
/**< maximum frequency in Hz of PWM signals at the beginning of start-up procedure */

#define HRTIM_MIN_PWM_FREQ_START_UP_HZ 120000
/**< minimum frequency in Hz of PWM signals at the end of start-up procedure for closed loop
operation if the control loop is not closed before */

#define HRTIM_MAX_PWM_FREQ_HZ 250000
/**< maximum frequency in Hz of PWM signals during closed loop regulation */

#define HRTIM_MIN_PWM_FREQ_HZ 120000
/**< minimum frequency in Hz of PWM signals during closed loop regulation
- WARNING: if f_min < 140626 Hz, extended PID must be used because period_max > S16_MAX */
```

In the same file, the time duration of start-up procedure is also defined:

```
#define DSMPS_STARTUP_TIME_DURATION_mS((uint16_t)500)
/**< duration in ms of DSMPS_START state*/
```

If this start-up time is elapsed without reaching the target value (e.g., if there is a short on the output), the start-up fault is triggered and the system is put in safe state.

The start-up procedure can be disabled by commenting the following line in in LLC_control_param.h file:

```
#define FREQUENCY_RAMP_UP
/**< if defined a a frequency start-up is performed in START state*/
```

If defined, frequency start-up is also performed in open loop mode. In this case, the initial switching frequency is always defined by HRTIM_MAX_PWM_FREQ_START_UP_HZ (380 kHz) and scales down to HRTIM_MAX_PWM_FREQ_HZ (250 kHz), which is both the maximum and default switching frequency in open loop mode. After start-up, it is possible to change open loop switching frequency.

2.7 Burst mode

The STSW-DPSLLCK1 application firmware package implements two different burst modes: the first one is related only to output voltage and is triggered when its value rises above a certain threshold during fast load transients. When this event is detected, PWM outputs are temporarily disabled until output voltage falls down to a second threshold. During the turn-off period, the control loop is disabled to avoid saturation of the accumulator (integral part of PID regulator). The burst mode threshold is set at a value lower than output overvoltage protection value and is defined, together with the re-activation threshold, in the Fault_Processing.c file:

```
/* Burst mode on bus voltage thresholds */
```

```
#define OUT_VOLTAGE_BURST_MODE_TH_H OUT_VOLT_ADC_VALUE(50)
/**< max voltage for burst mode activation [V] */
```

```
#define OUT_VOLTAGE_BURST_MODE_TH_L OUT_VOLT_ADC_VALUE(47)
/**< min voltage for burst mode de-activation [V] */
```

The second burst mode type is the light load burst mode, a common procedure in resonant converters.

The burst mode is implemented through a special feature of the HRTIM, setting burst idle and period duration, both expressed in number of PWM periods (refer to STM32F334 reference manual for further information).

Default light load burst mode parameters are defined in LLC_control_param.h file:

```
/*-- Burst Mode control ------------------------------*/
```

```
#define BURST_MODE_IDLE_DURATION_RANGE1 36
/**< if defined LIGHT_LOAD_BURST_MODE, BURST_MODE_IDLE_DURATION_RANGE1 + 1
is the number of HRTIM periods that compose the burst idle duration
when the burst mode is active and output current is in range1 */
```

```
#define BURST_MODE_PERIOD_RANGE1 39
/**< if defined LIGHT_LOAD_BURST_MODE, BURST_MODE_PERIOD_RANGE1 +1 is the
number of HRTIM periods that compose the burst period (idle+ run) when
the burst mode is active and output current is in range 1 */
```

Default burst idle duration is 36, whereas the default burst period is 39, this means that inside a burst period of 40 PWM cycles, 37 cycles are skipped and only 3 cycles are active (run duration).

Light load burst mode is enabled when output current goes below 0.8 A and disabled when the current goes above 1.3 A. More than one burst mode range can be defined according to output current measurements and desired active pulses, but only the first range is active by default.

Both burst modes are enabled by default, but they can be disabled by commenting the following lines in LLC_control_param.h file:

```
#define OUT_VOLTAGE_BURST_MODE
/**< if defined PWM signals are stopped if output voltage goes above
defined thresholds and re-enabled with hysteresys */
```

```
#define LIGHT_LOAD_BURST_MODE
/**< if defined PWM signals are in burst for light loads */
```

2.8 Faults and protections

The STSW-DPSLLCK1 application firmware package includes several protection functions that can be enabled or disabled by the user in LLC_control_param.h file uncommenting or commenting the corresponding line:

```
#define OVERCURRENT_PROTECTION
#define OUT_OVERCURRENT_PROTECTION
#define OVERTEMPERATURE_PROTECTION
#define INPUT_UNDER_OVER_VOLTAGE_PROTECTION
//#define VOUT_ANALOG_WATCHDOG_ENABLED
#define VOUT_UNDERVOLTAGE_PROTECTION
#define START_UP_FAILED_ENABLED
```

Each fault is identified by an error code defined in the DSMPS_type.c file. These codes present an “1” in a different bit position: in this way, the fault variable (DCDC_System_Fault, defined in the Fault_Processing.c file) is given by the logic OR operation of codes corresponding to all faults occurred. Some of these faults will be automatically cleared once the fault condition disappears (like input undervoltage), then the fault variable is updated by setting at “0” the corresponding bit. Other faults are not recoverable by choice, even if this setting (recoverable or not) can be modified by commenting the corresponding line in the Fault_Processing.c file:

```
#define OVER_TEMP_FAULT_NOT_RECOVERABLE
/**< if defined the over-temperature fault is not recoverable (the board must be powered
off) */
```

```
#define OVER_VOLTAGE_OUT_FAULT_NOT_RECOVERABLE
/**< if defined the output over-voltage fault is not recoverable (the board must be powered
off) */
```

```
#define UNDER_VOLTAGE_OUT_FAULT_NOT_RECOVERABLE
/**< if defined the output under-voltage fault is not recoverable (the board must be powered
off) */
```

However, overcurrent and output overcurrent protections are always not recoverable and they require either a system reset to be cleared, or a request by the communication interface.

Once a fault occurred, the system is put in STOP state and then in FAULT state, after disabling outputs, while the last fault occurred is stored in the DCDC_Last_System_Fault variable. The fault can be detected either via UI, reading the variable (during debugging), or by the blinking fault LED (red) on the control board. Only the first detected fault is signalled by LED blinking according to a different number of blinks and a different timing for each type of fault. The blinking sequence is then repeated after a pause given by the following definition in the DSMPS_type.h file:

```
#define LED_BLINK_REPETITION_PERIOD_MS  3000
/**< repetition period in ms between a blinking series and the next one */
```

The fault LED blinks with a 50% duty (ON and OFF) with a duration that can be short or long according to the following definitions:

```
#define LED_BLINK_PERIOD_LONG_MS 500
/**< LED on/off long period in ms */
```

```
#define LED_BLINK_PERIOD_SHORT_MS 250
/**< LED on/off short period in ms */
```

By default, slow blinks are related to voltage errors (input and output undervoltage and overvoltage), whereas the fast blinks are related to other errors (overcurrent, overtemperature, etc.).

The following table shows the different blinking configurations for each fault.

Table 3. Fault description

Error name	Code	Condition	Number of LED Blinks	Blinking speed	Recoverable
DCDC_NO_ERROR	0x0000	-	-	-	-
DCDC_OUT_OVER_VOLT_ERROR	0x0001	Vout > 56 V	3	slow	N
DCDC_OUT_UNDER_VOLT_ERROR	0x0002	Vout < 35 V	2	slow	N
DCDC_IN_OVER_VOLT_ERROR	0x0004	Vin > 435 V	4	slow	Y
DCDC_IN_UNDER_VOLT_ERROR	0x0008	Vin < 370 V	5	slow	Y
DCDC_OVER_CURRENT_ERROR	0x0010	Ires > Ires_max	2	fast	N
DCDC_OUT_OVER_CURRENT_ERROR	0x0020	Iout > 63 A	3	fast	N
DCDC_OVER_TEMP_ERROR	0x0040	T > 55 °C	4	fast	N
DCDC_STARTUP_FAILED_ERROR	0x0080	Vout < 47 V && Tramp > 500 ms	6	slow	N
DCDC_PRIMARY_SIDE_ERROR (from PFC if any)	0x0100	Error sent by UART communication (not enabled)	5	fast	Y
DCDC_COMMUNICATION_ERROR	0xE000	UI UART communication failed (not enabled)	6	fast	Y

Thresholds and hysteresis for each fault are defined in the Fault_Processing.c file:
/* - Fault and burst mode thresholds - for nominal values see LLC_Control_Parameters.h file - */ /* Out Voltage thresholds */

```
#define OUT_VOLTAGE_MAX_H OUT_VOLT_ADC_VALUE(56)
/**< max voltage for output overvoltage fault detection [V]
- if VOUT_ANALOG_WATCHDOG_ENABLED is not defined,otherwise
the WDG threshold is WDG_OUT_VOLTAGE_MAX in LLC_Init_Perif.c file */
```

```
#define OUT_VOLTAGE_HYSTERESIS OUT_VOLT_ADC_VALUE(4)
/**< hysteresis for output under/over voltage fault detection [V] */
```

```
#define OUT_VOLTAGE_MAX_L (OUT_VOLTAGE_MAX_H - OUT_VOLTAGE_HYSTERESIS)
 /**< min voltage for output overvoltage fault cancellation [V] */
```

```
#define OUT_VOLTAGE_MIN_L OUT_VOLT_ADC_VALUE(35)
/**< min voltage for output undervoltage fault detection [V] */
```

```
#define OUT_VOLTAGE_MIN_H (OUT_VOLTAGE_MIN_L + OUT_VOLTAGE_HYSTERESIS)
/**< max voltage for output undervoltage fault cancellation [V] */
```

```
/* Input Voltage thresholds */
```

```
#define IN_VOLTAGE_MAX_H IN_VOLT_ADC_VALUE(435)
/**< max voltage for input overvoltage fault detection [V] */
```

```
#define IN_VOLTAGE_HYSTERESIS IN_VOLT_ADC_VALUE(12)
/**< hysteresis for input under/over voltage fault detection [V] */
```

```
#define IN_VOLTAGE_MAX_L (IN_VOLTAGE_MAX_H - IN_VOLTAGE_HYSTERESIS)
/**< min voltage for input overvoltage fault cancellation [V] */
```

```
#define IN_VOLTAGE_MIN_L IN_VOLT_ADC_VALUE(360)
/**< min voltage for input undervoltage fault detection [V] */
```

```
#define IN_VOLTAGE_MIN_H (IN_VOLTAGE_MIN_L + IN_VOLTAGE_HYSTERESIS)
/**< max voltage for input undervoltage fault cancellation [V] */
```

```
/* Burst mode on bus voltage thresholds */
```

```
#define OUT_VOLTAGE_BURST_MODE_TH_H OUT_VOLT_ADC_VALUE(50)
/**< max voltage for burst mode activation [V] */
```

```
#define OUT_VOLTAGE_BURST_MODE_TH_L OUT_VOLT_ADC_VALUE(47)
 /**< min voltage for burst mode de-activation [V] */
```

```
/* Temperature thresholds */
```

```
#define TEMPERATURE_TH_H ((uint16_t)55)
/**< max temperature for over-temperature fault detection (higher threshold) [°C] */
```

```
#define TEMPERATURE_HYSTERESIS ((uint16_t)5)
/**< hysteresis for over-temperature fault detection [°C] */
```

```
#define TEMPERATURE_TH_L (TEMPERATURE_TH_H - TEMPERATURE_HYSTERESIS)
/**< min temperature for over-temperature fault cancellation [°C] */
```

```
/* Out current threshold */
```

```
#define OUT_CURRENT_MAX HALL_IC_AMP2ADC_VALUE(62)
```

All fault checks are performed in FLT_FaultCheck() function, except for fast overcurrent protection and AWD (if enabled), which is called inside the while(1) loop in the main.

Overcurrent protection is a based on resonant current sensed signal. The resonant current is sensed by a current transformer (CT) and then rectified by small signal fast diodes; the signal is sent to the MCU and then compared by an internal comparator to a fixed threshold set on a DAC channel. If the sensed signal rises above the defined threshold, the HRTIM fault is triggered. This means that PWM outputs are automatically put in a safe state by the hardware and HRTIM1_FLT_IRQHandler interrupt routine is also called to assert the corresponding fault, put the state machine in STOP state and then in FAULT state. Moreover, the fault event is filtered by a digital filter set in HRTIM_config() function to prevent false triggers due to commutation noise. The overcurrent threshold, expressed in 12-bit format, is defined in the LLC_control_param.h file:

```
#define DAC_OVERCURRENT_COMP_THRESHOLD ((uint16_t) 3850)
```

This protection may be triggered when a very large load change is applied.

2.9 PWM modulation

To generate PWM driving signals and trigger ADC acquisitions, the high resolution timer has been used. HRTIM has a modular architecture with several timer units (master timer and Timer A to Timer E) and different external events available; this allows the generation of complex control signals to address most power conversion topologies (for further information, refer to RM0364). To drive both primary side and secondary side switches, different timers have to be synchronized; for this purpose, Timer A and Timer B are used to drive high voltage power MOSFETs, whereas Timer C and Timer D are used to drive SR power MOSFETs and the master timer is used for synchronization and ADC trigger generation.

The primary side switches are driven by a 50% duty cycle and a proper dead time defined in the HRTIM_pwm_config_param.h file, automatically kept constant by the peripheral even if switching frequency changes, unless the user changes it during operation.

The HRTIM half mode, specifically designed for resonant converters, allows the duty cycle to be automatically forced to a half period value when a new period is programmed. By writing HRTIM_PERxR register, Compare 1 value (HRTIM_CMP1xR) is automatically updated to the HRTIM_PERxR/2 value. Even if primary side MOSFETs are logically driven two-by-two (LS1 with HS2 and HS1 with LS2), four HRTIM outputs are used as the gate drivers placed in the power board require an inverted polarity for low side signal; the polarity of each PWM output can be changed in the HRTIM_pwm_config_param.h file:

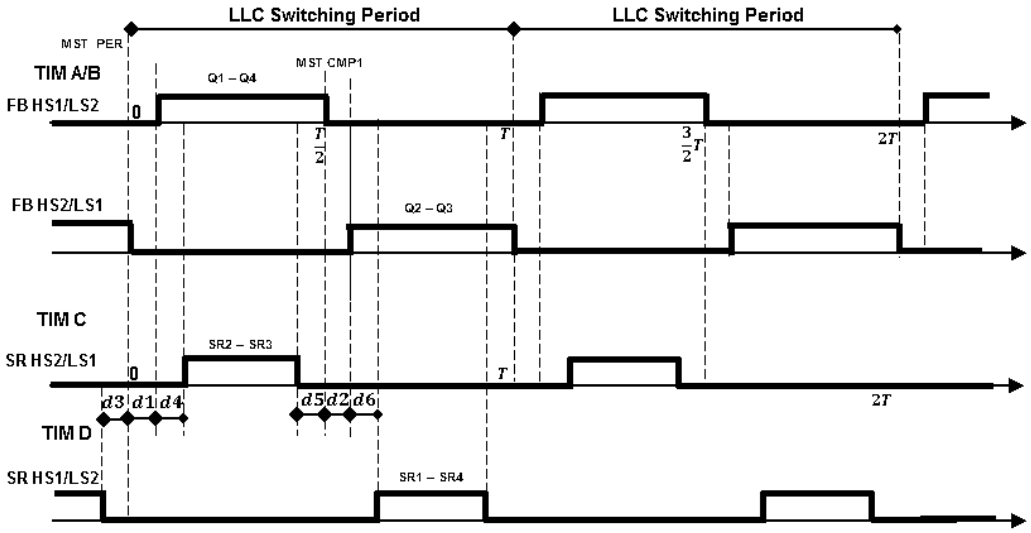
```
/* -- PWM out polarity */

#define HRTIM_PWM_FB_HIGHSIDE_OUTPOLARITY HRTIM_OUTPUTPOLARITY_HIGH

#define HRTIM_PWM_FB_LOWSIDE_OUTPOLARITY HRTIM_OUTPUTPOLARITY_LOW

#define HRTIM_PWM_SR_HIGHSIDE_OUTPOLARITY HRTIM_OUTPUTPOLARITY_HIGH

#define HRTIM_PWM_SR_LOWSIDE_OUTPOLARITY HRTIM_OUTPUTPOLARITY_LOW
```

In the same way, to drive SR power MOSFETs, four different PWM outputs (defined in the LLC_board_config_param.h file) have been used.

Whereas the driving of high voltage switches requires only the update of the period register, low voltage power MOSFETs are not driven by a 50% duty cycle, then two compare registers (for rising and falling edges) have to be written for each waveform, according to SR control logic. The transformer output voltage is rectified by output stage. Instead of using diodes, the rectification is performed by SR power MOSFETs that are turned on when the corresponding body diode is forward biased, considering also safe delays, as shown in the figure below.

Figure 7. PWM signals timing

- d1: dead time rising FB
- d2: dead time falling FB
- d3: SR delay falling2
- d4: SR delay rising1
- d5: SR delay falling1
- d6: SR delay rising2



In the above figure, d1 and d2 are the dead times (rising and falling of primary side power MOSFETs. Rising and falling dead times are equal and their default value is defined in HRTIM_pwm_config_param.h

```
/* -- Dead time calculation ---------------------------------------------- */
```

```
#define DEAD_TIME_RISING_NS ((uint16_t)600)
/**< initial or default dead time of rising edge in ns */
```

```
#define DEAD_TIME_FALLING_NS ((uint16_t)600)
/**< initial or default dead time of falling edge in ns */
```

```
#define DEAD_TIME_MAX_NS ((uint16_t)800)
/**< max dead time for both rising and falling edges in ns */
```

```
#define DEAD_TIME_MIN_NS ((uint16_t)200)
/**< min dead time for both rising and falling edges in ns */
```

The same file also defines allowable minimum and maximum values for saturation limits if the dead time is changed during operation (avoiding cross-conduction.

Delays d3 to d6 represent SR safe delays with respect to primary side waveforms: delay rising1, delay rising2, delay falling 1 and delay falling 2. Rising delays are always fixed, falling delays can change if adaptive SR is enabled, otherwise they are also fixed.

Default, minimum and maximum values for SR delays are defined in the LLC_control_param.h file.

```
/* -- Synchronous Rectification min values -- */

#define SYNCH_RECT_DELAY_RISING1_MIN_NS ((int16_t)0)
/**< min delay in ns between FB PWM and SR1 PWM rising edges in ns
- only for adaptive SR and on-fly delay setting */

#define SYNCH_RECT_DELAY_RISING2_MIN_NS ((int16_t)0)
/**< min delay in ns between FB PWM and SR2 PWM rising edges in ns
- only for adaptive SR and on-fly delay setting */

#define SYNCH_RECT_DELAY_FALLING1_MIN_NS ((int16_t)50)
/**< min delay in ns between FB PWM and SR1 PWM falling edges in ns
- only for an adaptive SR */

#define SYNCH_RECT_DELAY_FALLING2_MIN_NS ((int16_t)50)
/**< min delay in ns between FB PWM and SR2 PWM falling edges in ns
- only for an adaptive SR WARNING: Consider that auto-delayed Compare is only valid from the
capture up to the period event (RM pag 633)*/

/* -- Synchronous Rectification max values -- */

#define SYNCH_RECT_DELAY_RISING1_MAX_NS ((int16_t)600)
/**< max delay in ns between FB PWM and SR1 PWM rising edges in ns
- only for an adaptive SR */

#define SYNCH_RECT_DELAY_FALLING1_MAX_NS ((int16_t)600)
/**< max delay in ns between FB PWM and SR1 PWM rising edges in ns
- only for an adaptive SR */

#define SYNCH_RECT_DELAY_RISING2_MAX_NS ((int16_t)600)
/**< max delay in ns between FB PWM and SR2 PWM rising edges in ns
- only for an adaptive SR */

#define SYNCH_RECT_DELAY_FALLING2_MAX_NS ((int16_t)600)
/**< max delay in ns between FB PWM and SR2 PWM rising edges in ns
- only for an adaptive SR */

/* -- Synchronous Rectification init values -- */

#define SYNCH_RECT_DELAY_RISING1_INIT_NS ((int16_t)(250))
/**< init rising delay for SR1 in ns */

#define SYNCH_RECT_DELAY_FALLING1_INIT_NS SYNCH_RECT_DELAY_FALLING1_MAX_NS
/**< init falling delay for SR1 in ns */

#define SYNCH_RECT_DELAY_RISING2_INIT_NS ((int16_t)(250))
/**< init rising delay for SR2 in ns */

#define SYNCH_RECT_DELAY_FALLING2_INIT_NS SYNCH_RECT_DELAY_FALLING2_MAX_NS
/**< init falling delay for SR2 in ns */
```

The user should be careful when changing these values to avoid power MOSFET cross-conduction and to avoid loosing some SR PWM edges. If a calculated value for a compare register, which depends on both fixed dead time and SR delays, is greater than actual PWM period (given by control loop), the SR power MOSFETs will remain permanently ON or OFF.

If adaptive SR is enabled, SR power MOSFET turn-off can occur either on a HRTIM compare event or on a comparator trigger (with its proper blanking time).

Table 4. Registers and events used for PWM generation

Signal	Edge	Register/Event
FB HS1/LS2	RISING	TIMA_PER (TIMB_PER), dead time rising is automatically inserted
	FALLING	TIMA_CMP1 (TIMB_CMP1), automatically written in half mode
FB HS2/LS1 (complementary waveform)	RISING	TIMA_CMP1 (TIMB_CMP1), dead time falling automatically inserted
	FALLING	TIMA_PER (TIMB_PER)
SR HS2/LS1	RISING	TIMC_CMP2
	FALLING	TIMC_CMP1 or COMP2 event (Adaptive SR)
	BLANKING TURN-OFF (ADAPTIVE SR)	TIMC_CMP3
SR HS1/LS2	RISING	TIMD_CMP2
	FALLING	TIMD_CMP1 or COMP4 event (Adaptive SR)
	BLANKING TURN-OFF (ADAPTIVE SR)	TIMD_CMP3

Each register is updated with the values shown in the following table.

Table 5. Values for register update

Register	Value
TIMA_PER (TIMB_PER)	last PWM switching period (Ts) given by control loop
TIMC_CMP2	Dead time rising (d1) + SR delay rising1 (d4)
TIMC_CMP1	Ts/2 - SR delay falling1 (d5)
TIMC_CMP3	TIMC_CMP2 + blanking window duration
TIMD_CMP2	Ts/2 + dead time falling (d2) + SR delay rising2 (d6)
TIMD_CMP1	Ts - SR delay falling2 (d3)
TIMD_CMP3	TIMD_CMP2 + blanking window duration

When the control loop provides a new switching period, HRTIM registers have to be updated. To prevent mismatches, the update of used timers is disabled until all registers are written, so they are all active together at next counter roll-over event as pre-load is enabled. Different functions are defined in LLC_PWMnCurrVoltFdbk.c to update registers used to drive primary side power MOSFETs, secondary side power MOSFETs, or all together.

2.10 ADC acquisitions and triggers

Several ADC acquisitions are necessary to perform output regulation and enable protection features. Both ADC1 and ADC2 are used to acquire analog signals: output voltage, input voltage, output current and temperature (voltage across a NTC resistor). These acquisitions are divided into two different regular sequences triggered by the HRTIM master timer (Compare 2 and Compare 3 registers) to avoid commutation noise. Conversion results are then transferred by two DMA channels to DCDC_MeasureStruct variable, whose type is defined in the LLC_type.h file. If adaptive SR is enabled, two injected measurements (one for each ADC peripheral) of SR Vds voltages are also performed after a specified delay from SR MOSFETs turn-off. These measurements are used by the adaptive SR algorithm.

ADC trigger delays for both regular and injected conversions are defined in the LLC_control_param.h file:

```
/*-- ADC trigger point ------------------------------------------------------*/

#define ADCx_REGULAR_TRIGGER_POINT_NS ((uint16_t)(1300 + DEAD_TIME_RISING_NS))
/**< ADCx regular aquisition trigger point in ns respect to the beginning of HRTIM period
(rollover event) */

#define ADCy_REGULAR_TRIGGER_POINT_NS ((uint16_t)(500 + DEAD_TIME_FALLING_NS))
/**< ADCy regular aquisition trigger point in ns respect to half HRTIM period */

#define ADCx_INJECTED_TRIGGER_POINT_NS ((uint16_t)(250))
/**< ADCx injected aquisition trigger point in ns after PWM SR1 turn-off */

#define ADCy_INJECTED_TRIGGER_POINT_NS ((uint16_t)(250))
/**< ADCy injected aquisition trigger point in ns after PWM SR2 turn-off */
```

2.11 Synchronous rectification

In several power conversion systems, such as AC-DC and DC-DC converters, the output diode (or diodes) rectifier can be replaced by a power MOSFETs, driven by a control logic, to improve the converter efficiency through a technique called synchronous rectification (SR). Thanks to SR, conduction losses are significantly reduced as the output current flows through the power MOSFETs channel instead of the rectification diode; the power loss is then decreased from:

$$P_{loss\_diode} = V_d \cdot I_{out}$$

to:

$$P_{loss\_MOSFET} = R_{ds\_on} \cdot I_{out}^2$$

where R_{ds_on} is very low for SR power MOSFETs.

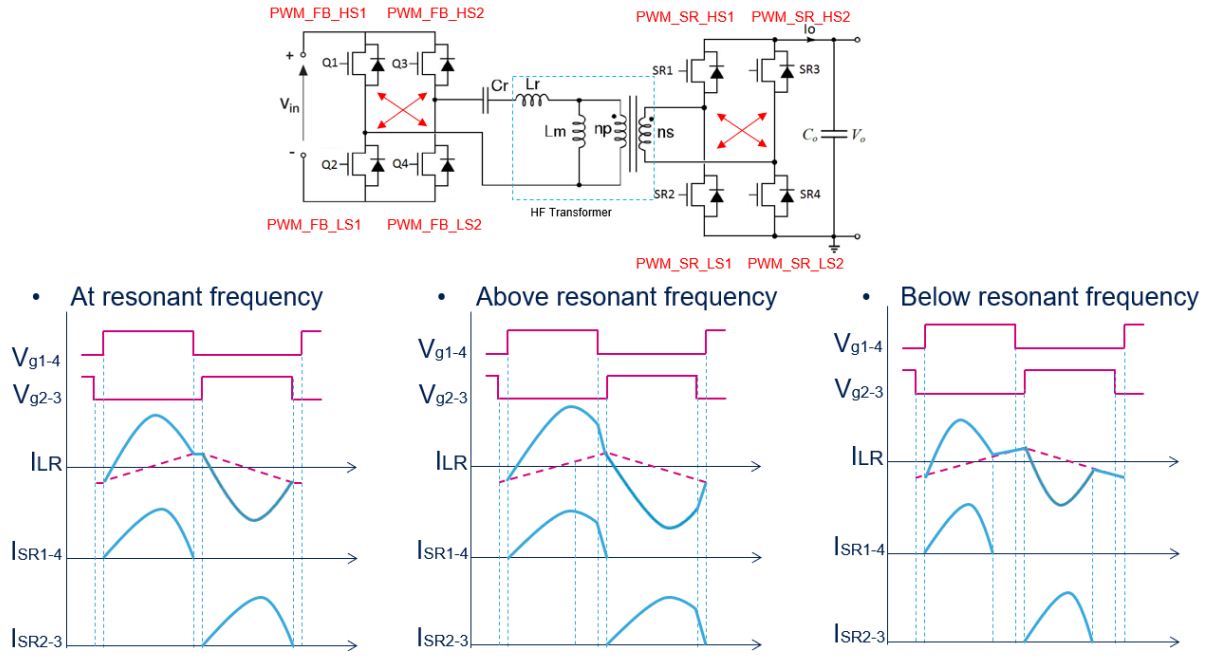
The power converter works even when the SR power MOSFETs are not driven because the rectification is naturally performed by the internal body diode, but suffers worse performance, whereas driving the power MOSFETs when the body diode is forward biased, the system efficiency can increase by 3% - 4%.

A digital implementation of the synchronous rectification allows a reduction in the number of components, introduces more flexibility in control algorithm design and guarantees a higher tolerance to noises.

In LLC topology, the conduction time of SR devices depends on operating frequency and is related to primary side driving signals. If the LLC converter works below the main resonant frequency, but still in the inductive region, SR devices conduct for less than half the PWM period because resonant current reaches magnetizing current before the end of switching semi-period, so conduction time should be adapted.

The figure below shows the conduction time of rectification devices according to switching frequency at resonant frequency, above the resonance and below the resonance.

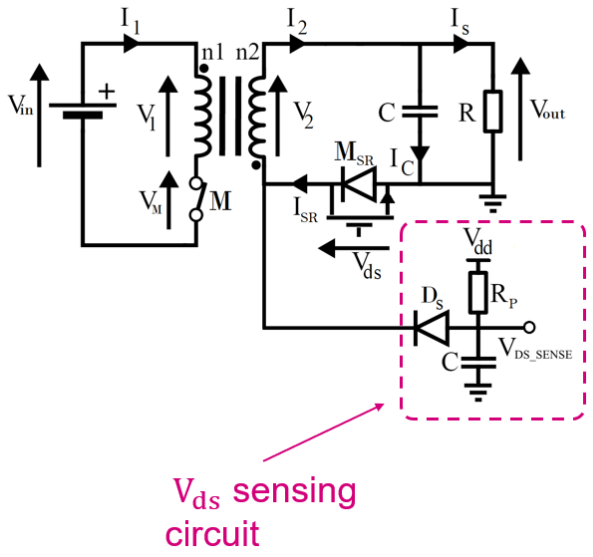
Figure 8. LLC: Conduction of SR devices



In the STSW-DPSLLCK1 application firmware package, there are two independent signals (managed by TIMC and TIMD, respectively) for each diagonal of the rectification stage, and the SR driving can be either with fixed delays (with respect to primary side signals) or adaptive ones. In the second case, SR rising delays are fixed and set at 250 ns to avoid MOSFETs turning on too early when body diodes can shortly conduct for capacitive currents. However, falling delays can be automatically tuned according to body diode conduction time,which is detected by acquiring the Vds voltage across low side SR MOSFETs.

The sensing network consists of a fast Schotthky diode and a pull-up resistor connected to the MCU supply voltage, as shown in the figure below.

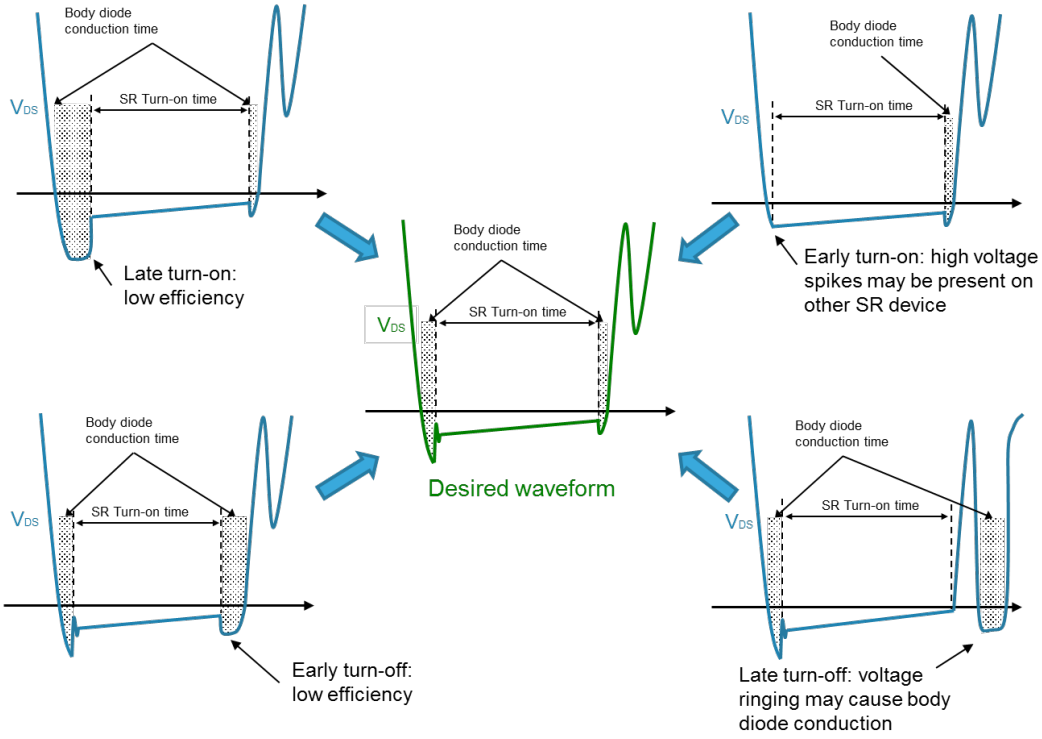
Figure 9. SR Vds sensing circuit in a flyback converter



It is important that this sensing network is applied only to low side power MOSFETs, whose source is connected to the microcontroller same ground. When the SR power MOSFET drain voltage is above the MCU supply voltage Vdd, the Schotthky diode is reverse biased and the sensed voltage is pulled up to Vdd. When drain voltage is below Vdd, the Schotthky diode is forward biased and the sensed voltage is equal to this voltage plus the diode voltage drop, generating a positive shift. The current during positive biasing is limited by the pull-up resistor and the diode voltage drop depends on this current. When the body diode is conducting, Vds becomes negative (forward biased diode), but when power MOSFET conducts Vds=Rds_on*I_out.

A safe dead time should be inserted before the turn-on and after the turn-off in which body diode conducts.

Figure 10. Typical and desired Vds waveforms for a flyback converter with SR



Adaptive SR technique optimizes dead times. If the generated PWM signal is turned-off too early, a positive current causes body diode conduction that lowers converter efficiency, so part of SR benefits are lost. If the PWM is turned-off too late, the power MOSFET is forced to conduct even if the diode is reverse biased, which can damage the power MOSFETs and cause its failure. Moreover, after the power MOSFETs turn-off, a voltage ringing of the Vds can force the body diode to conduct again, impacting efficiency.

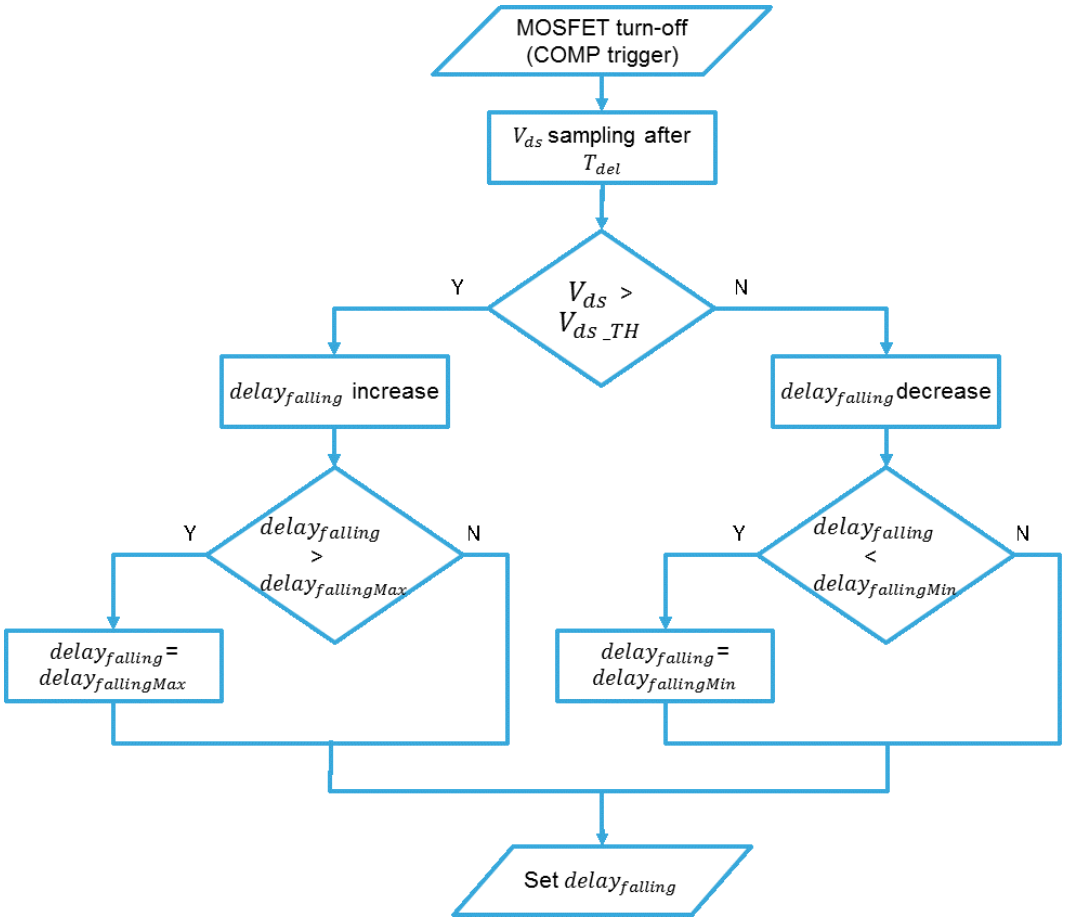
In the firmware package, early or late turn-off of SR MOSFET is detected using injected ADC acquisition after power MOSFET turn-off and the pulse duration is adjusted according to an ST patented adaptive algorithm. The ADC trigger is given by the HRTIM TIMC (and TIMD for the second SR diagonal) and Compare 4 register is set to autodelayed mode from Compare 1 register (which gives the power MOSFETs turn-off). This adds a fixed delay to a compare event or an external event, like the COMP trigger used as a second turn-off mechanism.

The autodelayed mode is a special feature of HRTIM. If the acquired value is below a preset threshold, the power MOSFET body diode is still conducting and the PWM is turned-off too early as the voltage drop on the power MOSFET is equal to the forward voltage of body diode (Figure 10 at the bottom-left). The conduction time can be increased (falling delay decreased) by a fixed quantity.

If the acquired value is above the preset threshold, the power MOSFET is forced to conduct even with a reverse current because the PWM is turned-off too late due to the fact that a small safe interval in which the diode conducts (little notch in Vds waveform) is not present and the Vds rises quickly (Figure 10 at the bottom-right). In this case, the conduction time can be decreased (falling delay increased) by a fixed quantity to turn off the power MOSFET earlier.

The sampling delay from PWM turn-off also provides the time duration of this safe body diode conduction interval. The adaptive algorithm flow chart is shown below, taking into consideration also the saturation limits.

Figure 11. **Adaptive SR algorithm flowchart**



In practice, to speed reaching the steady state, two couples of thresholds, with different incremental values, are defined in the LLC_control_param.h file:

```
/* -- Adaptive SR incremental/decremental values -- */

#define ADAPTIVE_SYNCH_RECT_INCREMENTAL_DELAY1_NS ((int16_t)2)
/**< incremental falling delay1 for adaptive SR in ns */

#define ADAPTIVE_SYNCH_RECT_INCREMENTAL_DELAY2_NS ((int16_t)5)
/**< incremental falling delay2 for adaptive SR in ns */

#define ADAPTIVE_SYNCH_RECT_DECREMENTAL_DELAY_NS ((int16_t)2)
/**< decremental falling delay for adaptive SR in ns */
```

```
/* -- Adaptive SR thresholds values */

#define ADAPTIVE_SYNCH_RECT_INCREMENTAL_VDS1_TH1_mV ((uint16_t)1000)
/**<ADC Vds1 threshold1 in mV to increase/decrease falling delay in adaptive SR */

#define ADAPTIVE_SYNCH_RECT_INCREMENTAL_VDS1_TH2_mV ((uint16_t)1200)
/**< ADC Vds1 threshold2 in mV to increase falling delay in adaptive SR */

#define ADAPTIVE_SYNCH_RECT_INCREMENTAL_VDS2_TH1_mV ((uint16_t)750)
/**< ADC Vds2 threshold1 in mV to increase/decrease falling delay in adaptive SR */

#define ADAPTIVE_SYNCH_RECT_INCREMENTAL_VDS2_TH2_mV ((uint16_t)850)
/**< ADC Vds2 threshold2 in mV to increase falling delay in adaptive SR */
```

The calculation of the new delays is performed in the CTR_AdaptiveSynchRectEdgeCalculation() function called inside the control loop interrupt routine.

To prevent current inversion, MOSFET turn-off can also be triggered by internal comparators (one for each SR diagonal) whose thresholds are set in specific DAC channels and defined in the LLC_control_param.h file:

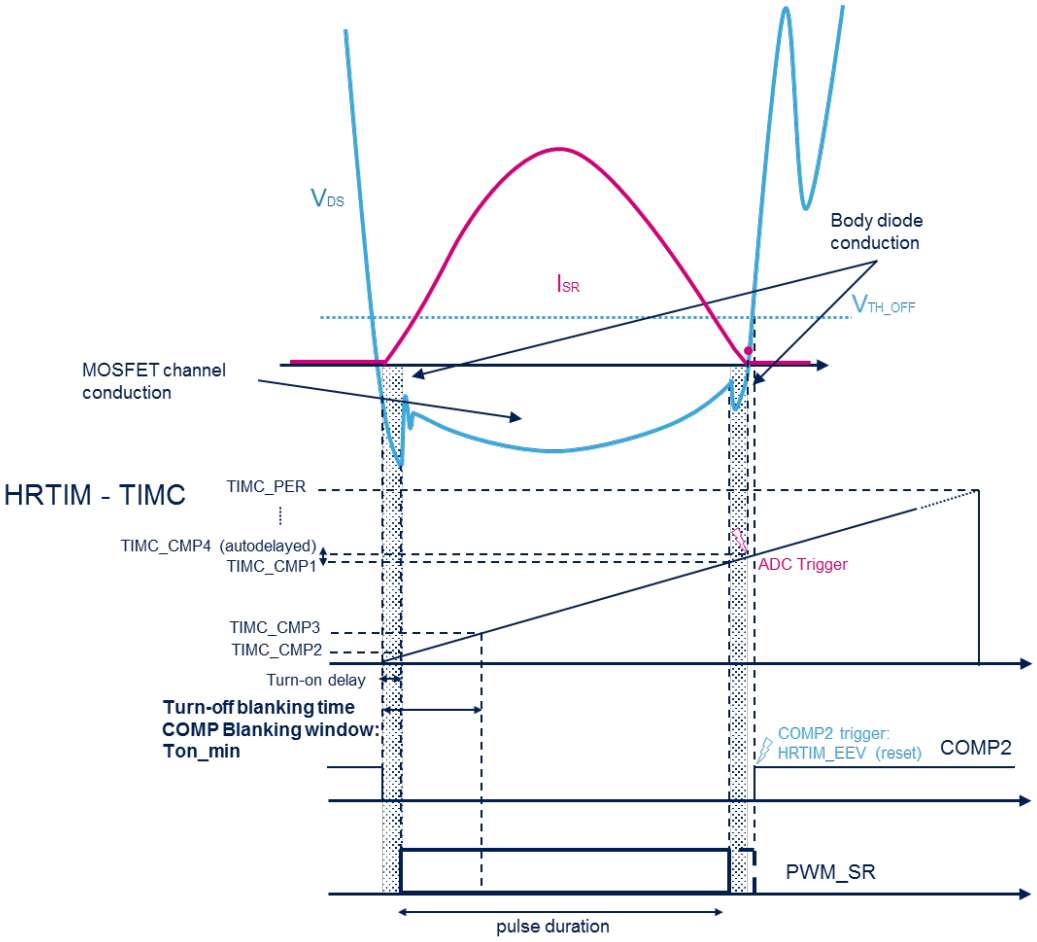
```
/*-- Turn-off COMP thresholds for adaptive SR --------------------------------*/

#define DAC_SR1_TURN_OFF_COMP_THRESHOLD_mV ((uint16_t) 900)
/**< COMP Vds1 threshold in mV on DAC to shut-down PWM in adaptive SR */

#define DAC_SR2_TURN_OFF_COMP_THRESHOLD_mV ((uint16_t) 800)
/**< COMP Vds2 threshold in mV on DAC to shut-down PWM in adaptive SR */
```

To avoid a false trigger of the internal comparator, due to commutation noise at MOSFET turn-on, a blanking window is inserted to prevent an early turn-off and ensure a minimum on-time. The blanking window duration is defined in the LLC_control_param.h file:

```
/*-- Blanking window duration for COMP turn-off in adaptive SR ---------------*/

#define SYNCH_RECT_TURN_OFF_BLANKING_WINDOW_NS ((int16_t)700)
/**< minimum turn-on time for SR1/SR2 signlas for adaptive SR in ns */
```

The timing diagram for SR PWM generation for one diagonal is shown in the figure below.

Figure 12. SR PWM generation: timing diagram



The above image shows how pulse generation starts after a delay given by the TIMC_CMP2 register, then the Vds rings due to MOSFET turn-on, but it is not immediately turned-off by the comparator trigger (threshold in dotted blue line along the top) because the threshold is high and TIMC_CMP3 ensures a minimum on-time (blanking window). TIMC_CMP1 gives the MOSFET turn-off, after 250 ns (TIMC_CMP4 in autodelayed mode from TIMC_CMP1) the ADC injected acquisition of Vds is triggered (the purple point on top) to perform adaptive SR. In this case, COMP has not been triggered since it is used as a protection system to guarantee MOSFET turn-off.

RELATED LINKS

AN2450: LLC resonant half-bridge converter design guideline

AN2644: An introduction to LLC resonant half-bridge converter

2.12 Fan driving

The power board cooling fan is supplied by a DC voltage, which, according to the jumper setting and power MOSFET driving, can be the 48 V output voltage, a lower voltage derived from it, or an external voltage.

If J4 jumper is disconnected, an external supply voltage can be directly applied to the fan using J5 connector. Otherwise, if J4 is connected, it is possible to reduce the 48 V applied to the fan by modulating the PWM signal driving the Q13 power MOSFET of the dedicated step-down converter.

The fan rotation speed is proportional to the voltage applied to it and therefore to the duty cycle of the PWM that drives the power MOSFET.

In low frequency tasks, the duty cycle of the PWM used to drive the power MOSFET is adjusted according to the load and the heat-sink temperature: if the temperature is above the higher threshold, the maximum duty cycle is applied, otherwise the duty cycle is proportional to the load.

Temperature and current thresholds, PWM frequency, and minimum and maximum duty cycle values are defined in the LLC_control_param.h file:

```
/*-- PWM parameter for FAN control ------------------------------*/

#define FAN_PWM_FREQUENCY 10000
/* PWM Frequency in Hz - min 367 Hz with prescaler = 2 - see FAN_TIM_Config() */

#define FAN_PWM_INIT_DUTY FAN_PWM_DUTY_LOW_SPEED
/**< init duty cycle for fan control */

#define FAN_PWM_DUTY_LOW_SPEED ((uint16_t)((30*FAN_PWM_TIM_PERIOD)/100))
/**< low speed duty cycle for fan control */

#define FAN_PWM_DUTY_HIGH_SPEED ((uint16_t)((97*FAN_PWM_TIM_PERIOD)/100))
/**< high speed duty cycle for fan control */

#define FAN_TEMPERATURE_THRESHOLD_H 43
/**< temperature high threshold (for hysteresis): if temperature is above the threshold, the
fan is driven at high speed, otherwise the speed depends on the load (if FAN_PWM_DRIVING is
defined) */

#define FAN_TEMPERATURE_THRESHOLD_L 38
/**< temperature low threshold (for hysteresis): if temperature is above the threshold, the
fan is driven at high speed, otherwise the speed depends on the load (if FAN_PWM_DRIVING is
defined) */

#define FAN_HIGH_LOAD_THRESHOLD HALL_IC_AMP2ADC_VALUE(40)
/**<output current threshold to activate max duty cycle for fan */

#define FAN_LOW_LOAD_THRESHOLD HALL_IC_AMP2ADC_VALUE(6)
/**<output current threshold to activate min duty cycle for fan */
```

2.13 Open loop mode and debugging

The STSW-DPSLLCK1 application firmware package can be customized for debugging purposes, both acting at compilation time and when the application is running.

To change the converter default behavior by enabling or disabling certain features, the LLC_control_param.h can be modified accordingly.

At board power-up, the default setting for power conversion starts as soon as an input voltage in the correct range is detected after the WAIT state timeout has elapsed, without waiting for a user command (by UI, or debugger). Initially, only low side power MOSFETs are turned-on to pre-charge driver bootstrap capacitors, and the soft-start routine sets the initial switching frequency quite high but it gradually decreases. To disable the start-up features, the following lines can be commented.

```
/* Start-up ------------------*/

#define START_WITHOUT_COMMAND
/**< if defined the power conversion starts without any command if there is no error,
otherwise a command from UI or debugger is needed */

#define FREQUENCY_RAMP_UP
/**< if defined a frequency start-up is performed in START state */
```

```
#define GATE_DRIVER_BOOTSTRAP_PRECHARGE
/**< if defined the bootstrap capacitors are precharged closing low side switches for a
defined time duration */
```

Other driving functions, such as synchronous rectification (both adaptive and with fixed delays), fan driving and burst mode, can be removed by commenting the following lines:

```
/* Protections and burst modes ------------*/
```

```
#define OUT_VOLTAGE_BURST_MODE
/**< if defined PWM signals are stopped if output voltage goes above defined thresholds and
re-enabled with hysteresis */
```

```
#define LIGHT_LOAD_BURST_MODE
/**< if defined PWM signals are in burst for light loads */
```

```
/* Additional driving -------*/
```

```
#define SYNCH_RECTIFICATION
/**< if defined synchronous rectification is enabled to reduce conduction losses
 - PWMs depend on hSR_DelayRising1-2 and hSR_DelayFalling1-2 variables */
```

```
#define ADAPTIVE_SYNCH_RECTIFICATION
**< if defined synchronous rectification is enabled to reduce conduction losses
- PWMs depend on COMP outputs and turn-off is adapted respect to VDS sensing /
- it must be define together with SYNCH_RECTIFICATION */
```

```
#define AUTOMATIC_SR_TURN_ON_OFF
/**< if defined SR is automatically turned on and off depending on SR_IOUT_TURN_ON_THRESHOLD
and SR_IOUT_TURN_OFF_THRESHOLD */
```

```
#define FAN_PWM_DRIVING
/**< if defined the FAN is driven by PWM, otherwise the FAN is driven as a GPIO */
```

These features can alternatively be disabled at the beginning (but can be enabled again by the user according to the following settings:

```
/* initial state of additional driving and burst if defined */
```

```
#define SYNCH_RECTIFICATION_INIT_ENABLED
/**< if defined synchronous rectification is initially enabled, the set can be changed
runtime */
```

```
#define ADAPTIVE_SYNCH_RECTIFICATION_INIT_ENABLED
/**< if defined adaptive synchronous rectification (with adaptive turn-off) is initially
enabled, the set can be changed runtime */
```

```
#define FAN_PWM_DRIVING_INIT_ENABLED
/**< if defined fan PWM driving is initially
        enabled, the set can be changed runtime */
```

```
#define LIGHT_LOAD_BURST_MODE_INIT_ENABLED
/**< if defined, together with LIGHT_LOAD_BURST_MODE, burst mode is set for low load
condition */
```

The open loop mode consists in setting a constant switching frequency without any output regulation. In this case, output undervoltage protection is automatically disabled as the voltage gain, given by resonant tank at the specified switching frequency, may not be enough to ensure an output voltage greater than the protection threshold. The user should be careful about limiting the output voltage (i.e., connecting an electronic load set in voltage mode) even if the output overvoltage protection is kept enabled.

To enable the open loop mode, the corresponding line can be uncommented.

```
/* Debug ---------------------*/

//#define OPEN_LOOP_MODE
/**< if defined SR delays and PWM frequency are initially set via
debugger or communication (operation mode can be changed runtime)
- OPEN LOOP operation, WARNING: output voltage MUST be clamped @48V */
```

The open loop enable setting can be changed, as well as the other configurations, via the debugger through the variable DCDC_ConfigParamStruct. This structure variable type is defined in the LLC_type.h file:

```
/**
  * @brief  Configuration Param Struct
  */
typedef struct
 {
   bool bConverterEnabled;
   bool bOpenLoopEnabled;
   bool bSREnabled;
   bool bAdaptiveSREnabled;
   bool bBurstModeEnabled;
   bool bFanPWMDrivingEnabled;
   uint32_t wOpenLoopFreq_Hz;
   uint16_t hDeadTimeFullBridge_ns;
   int16_t hFixedSRDelayRising1_ns;
   int16_t hFixedSRDelayFalling1_ns;
   int16_t hFixedSRDelayRising2_ns;
   int16_t hFixedSRDelayFalling2_ns;
   int16_t hRegulatorKpGain;
   int16_t hRegulatorKiGain;
   int16_t hRegulatorKdGain;
   bool bConfigurationChanged;
 } DCDC_ConfigParamStruct_t;
```

Each boolean setting can be changed by assigning a TRUE or FALSE value to the corresponding variable.

The open loop switching frequency, expressed in Hz, can also be changed by modifying wOpenLoopFreq_Hz field in the range ([120000; 250000] Hz). In the same way, dead time and SR fixed delays (expressed in ns and with the limits defined in LLC_control_param.h and HRTIM_pwm_config_param.h files) can be modified, as well as regulator gains.

When the configuration is changed, the bConfigurationChanged field has to be set to TRUE to load new values into the low frequency task and become effective.

Example

If the user wants to give the start command to the converter, comment START_WITHOUT_COMMAND, and the state machine remains in IDLE even when the input voltage reaches the correct range. To start the power conversion, bConverterEnabled field and bConfigurationChanged must be set to TRUE. To stop the power conversion bConverterEnabled field should be set to FALSE and bConfigurationChanged to TRUE to update the configuration.

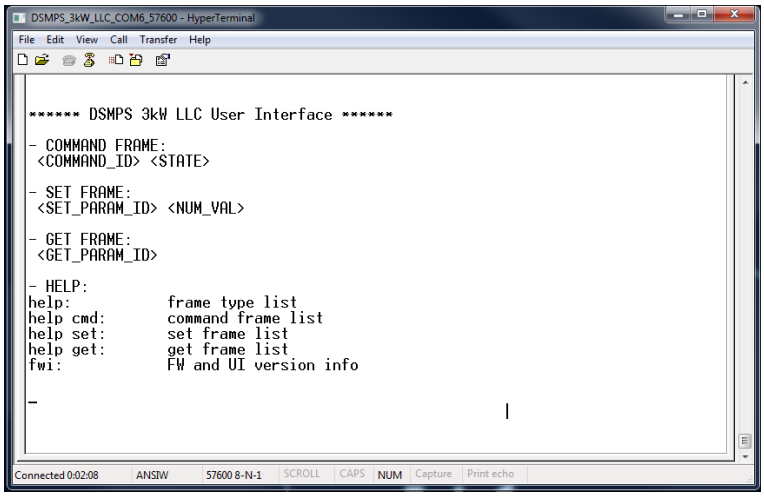
2.14 UI communication

The firmware configuration can also be changed at runtime using the UART communication interface. The UI is based on a simple RS-232 serial communication and any application that allows inserting and reading text (e.g., HyperTerminal or PuTTY) can be used. The communication should be set with the following parameters:

- Baud Rate: 57600
- Word Length bit: 8
- Stop bits: 1
- Parity: none
- Flow control: none

When the control board is powered, an initialization message is sent by UART to display help commands for each category.

Figure 13. UI interface: initialization message



To insert a new command the corresponding characters should be typed, followed by the ENTER key; in this way, the received string will be processed by the microcontroller. Commands are not case sensitive, blank spaces are ignored and the DEL key can be used to correct typing mistakes.

The frames that can be sent to the system are help frames, command frames, set frames and get frames.

The figures below show the help return messages for all frame categories.

Figure 14. UI: help command frame

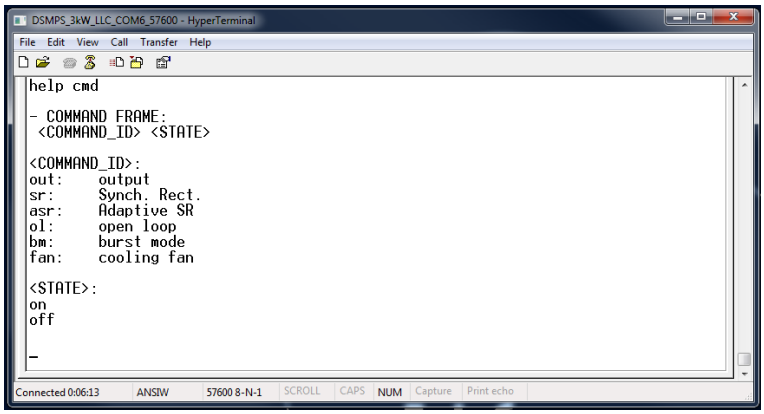


Figure 15. **UI:help set frame**

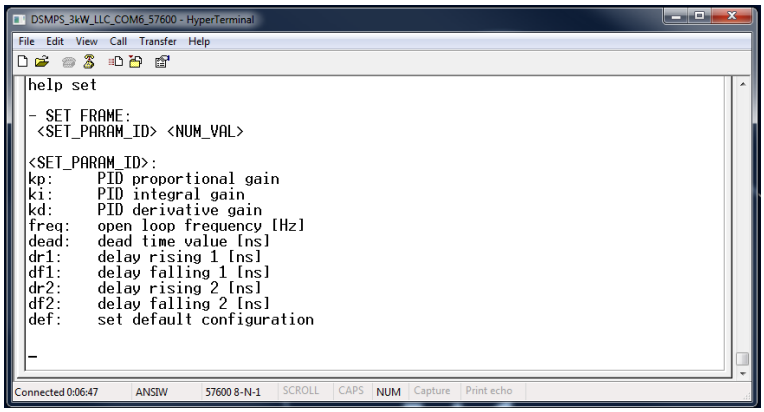
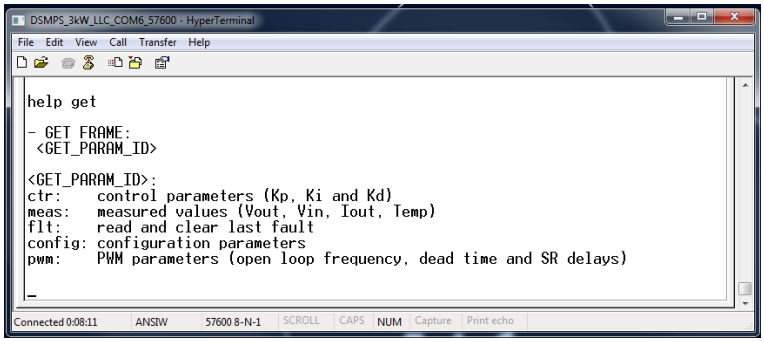


Figure 16. **UI: help get frame**



When a new command is received and recognized, it is processed by modifying the DCDC_ConfigParamStruct variable, then a return message is sent. Typing mistakes and out of boundary errors (for set commands) are recognized and displayed.

The following figures show examples for each frame type.

Figure 17. UI: command frame examples

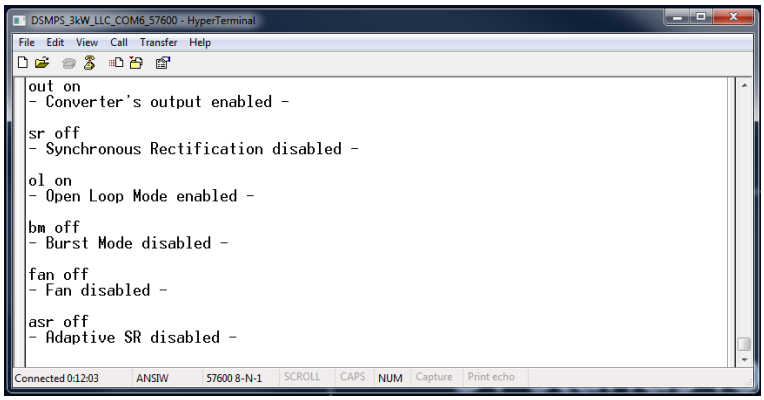


Figure 18. UI: set frame examples

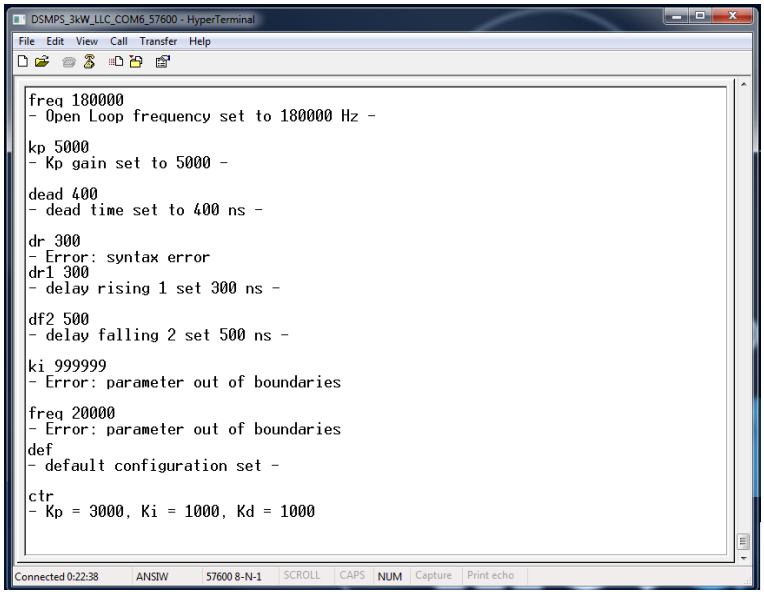
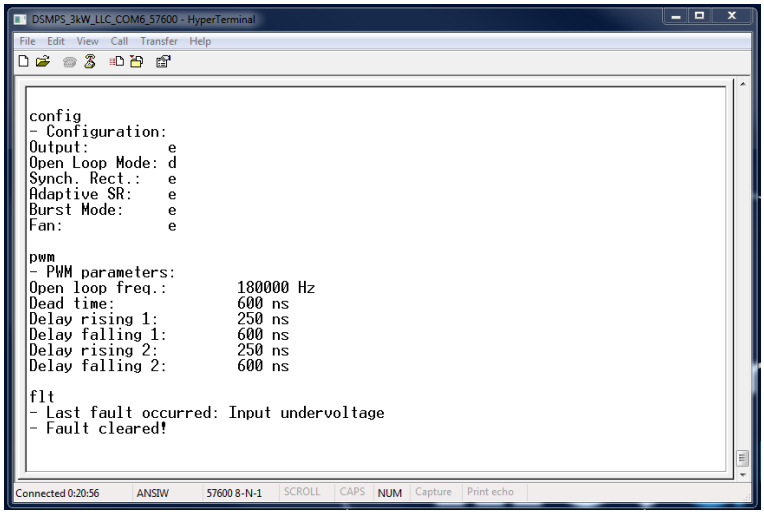


Figure 19. UI: get frame examples



Starting from firmware v1.7, the last configuration is stored into the Flash memory and re-loaded at power-on, as defined in the following line (in LLC_control_param.h file):

```
#define USE_FLASH_MEMORY
/**< if defined the actual configuration is written into flash memory and re-load at power-
on */
```

To recognize commands in the received string and process them, Lex and Yacc have been used.

Lex is a lexical analyzer that identifies meaningful tokens. Lex source is a table of regular expressions, written in a pseudo-C code (in lexer.l file for this application) in a simple way. The code used to recognize tokens is shown below.

```
%%
\n
[\t ]+    /* ignore white spaces */

^fwi|info {sprintf(retVal, "\n- Info: FW V1.7, UI V1.1 -\n\n\r");}
^help {sprintf(retVal, help_msg);}
^help[ ]?cmd {sprintf(retVal, help_cmd_msg);}
^help[ ]?set {sprintf(retVal, help_set_msg);}
^help[ ]?get {sprintf(retVal, help_get_msg);}

out|OUT|output {return TOKEN_OUT;}
SR|sr {return TOKEN_SR;}
ol|OL|(o|O)pen[ ]?(l|L)oop {return TOKEN_OL;}
asr|ASR|adaptive[ ]?(sr|SR) {return TOKEN_ADAPTIVE_SR;}
bm|BM|(b|B)urst[ ]?(m|M)ode {return TOKEN_BM;}
fan|FAN {return TOKEN_FAN;}

on|off yylval=!strcmp(yytext,"on"); return STATE;

kp|Kp {return TOKEN_KP_GAIN;}
ki|Ki {return TOKEN_KI_GAIN;}
kd|Kd {return TOKEN_KD_GAIN;}
freq|FREQ {return TOKEN_FREQ;}
dead|dead[ ]?time|dt|DT {return TOKEN_DEAD_TIME;}
(dr|DR)1 {return TOKEN_DR1;}
(dr|DR)2 {return TOKEN_DR2;}
(df|DF)1 {return TOKEN_DF1;}
(df|DF)2 {return TOKEN_DF2;}
def {return TOKEN_DEFAULT_CONFIG;}

ctr[l]?|CTR[L]? {return TOKEN_CTR_PARAM;}
meas|MEAS {return TOKEN_MEASURES;}
flt|FLT|fault|FAULT {return TOKEN_FAULT;}
cnf|CNF|config|CONFIG {return TOKEN_CONFIG_PARAM;}
pwm|PWM {return TOKEN_PWM_PARAM;}

[-+]?[0-9]+ {yylval=atoi(yytext); return NUMBER;}

[a-zA-Z]+ {return TOKEN_ERROR;}

%%
```

For example, if the user types a number, with or without sign (“+” or “-“), single digit or not, the token NUMBER is recognized and its value is returned in yylval variable. In the same way, typing “on” or “off”, the token STATE is detected and the corresponding value is stored in the yylval variable.

Yacc provides a general tool which helps describe the input to a computer program and how to write rules starting from the tokens found. For this application the rules are written in the parser.y file.

The following example shows the rule used to enable or disable the output, starting from TOKEN_OUT and STATE tokens and the value previously stored in yylval variable (“on” or “off”):

```
output_switch:
TOKEN_OUT STATE
{
if($2){
       DCDC_ConfigParamStruct.bConverterEnabled = TRUE;
       DCDC_ConfigParamStruct.bConfigurationChanged = TRUE;
       sprintf(retVal,"\n-Converter's output enabled -\n\n\r");}
else{
     DCDC_ConfigParamStruct.bConverterEnabled = FALSE;
     DCDC_ConfigParamStruct.bConfigurationChanged = TRUE;
     sprintf(retVal,"\n-Converter's output disabled -\n\n\r");
}
}
;
```

As already mentioned, the output enable is done by modifying the corresponding field in the DCDC_ConfigParamStruct variable and by setting the flag for the update, a return message is also sent.

A simple way to compile .l and .y files is by installing the latest versions of flex and bison. These files can then be compiled directly via a command prompt and the following instructions:

- set the path of lex and yacc files using cd directive; for example, if the firmware package is on your desktop:

  cd C:\Users\user name\Desktop\DSMPS 3kW - LLC STM32F334x\FullBridge_LLC_Project\EWARM\LexYacc
- compile .l file with directive: flex <filename>:

  flex lexer.l
- compile .y file with directive: bison -dy <filename>:

  bison -dy parser.y

Once these operations are performed, compiled files (lex.yy.c, y.tab.c and y.tab.h) are updated in the LexYacc folder and recognized in IAR Embedded Workbench toolchain, then the project can be compiled normally.

If the user interface is not used, you can free some space in the microcontroller memory by removing it as follows:

- undefine UI_COMMUNICATION in LLC_control_param.h
- exclude all C files in LexYacc folder from building (in IAR Embedded Workbench)
- remove UI_UART_Interface.c/.h files from building (in IAR Embedded Workbench)
- comment #include "UI_UART_Interface.h" in the stm32f3xx_hal_msp.c file
- compile the code again and flash it into the microcontroller

RELATED LINKS

UM2348: Getting started with the STEVAL-DPSLLCK1 evaluation kit for the 3 kW full bridge LLC digital power supply

For board schematics, visit the STEVAL-DPSLLCK1 product page on st.com.

3 System introduction

3.1 Getting started with the STEVAL-DPSLLCK1 evaluation kit

The 3 kW isolated full bridge LLC DC-DC resonant converter evaluation kit can convert 375 V to 425 V_{DC} input voltage to 48 V, 63 A maximum current. The kit (STEVAL-DPSLLCK1) consists of a power board, a control board and an adapter board. The STEVAL-DPS334C1 control kit is composed of the control board and the adapter board.

Figure 20. STEVAL-DPSLLCK1 evaluation kit

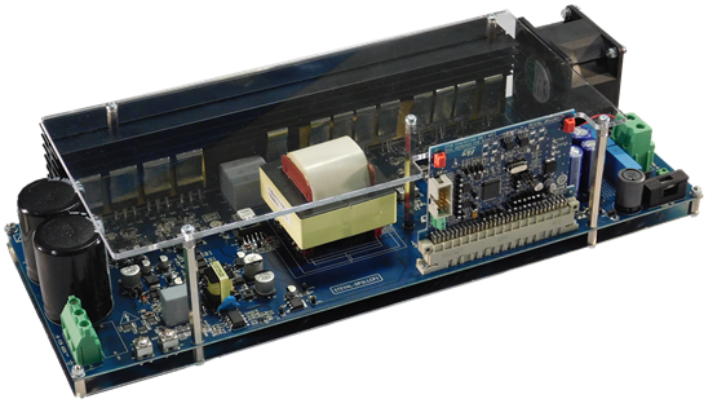


Table 6. STEVAL-DPSLLCK1 board specifications

Parameter	Value
Input DC voltage	375 to 425 V
Output voltage	48 V
Max output current	62.5 A
Output power	3000 W
Closed loop switching frequency	120 up to 250 kHz
Start-up switching frequency	380 kHz
Resonance frequency	175 kHz
HF transformer isolation	4 kV
Peak efficiency	95.3% @ 400 V 50% of load
Cooling	Forced air with fan speed modulation

3.2 General precaution

Danger:

During assembly and operation, the 3-kW full bridge LLC digital power supply poses several inherent hazards, including bare wires, rotating fan components and hot surfaces. There is danger of serious personal injury and damage to property if the DC-DC converter or its components are not used or installed correctly.

All operations involving transportation, installation, use and maintenance must be performed by skilled technical personnel able to understand and implement national accident prevention regulations. For the purposes of these basic safety instructions, “skilled technical personnel” are suitably qualified people who are familiar with the installation, use and maintenance of power electronic systems.

3.3 Electrical connection

Important:
The electrical installation shall be completed in accordance with the appropriate requirements (for example, cross-sectional areas of conductors, fusing, and GND connections). The kit is intended for evaluation purposes only. Supply the STEVAL-DPSLLCK1 only with a DC-DC source lab power supply.

Danger:
Do not touch the boards immediately after disconnection from the voltage supply as several parts and power terminals may contain energized capacitors that need time to discharge. Do not touch the boards after disconnection from the voltage supply as several parts like heat sinks and transformers may still be very hot.

Important:
Always use the STEVAL-DPSLLCK1 with the Plexiglass provided with the kit. Do not use the kit without the aluminum plate attached under the PCB. Always connect the earth ground connection on the input connector before you turn on the board.

3.4 Board cable connections

1. Connect the programmable DC voltage source to J2 with cables, or to B1 and B2 with bus bars. The output load must be connected to J3 connector with a cable with appropriate cross-section to carry the desired load current (63 A max).

Note: Always connect the earth to J2 connector.

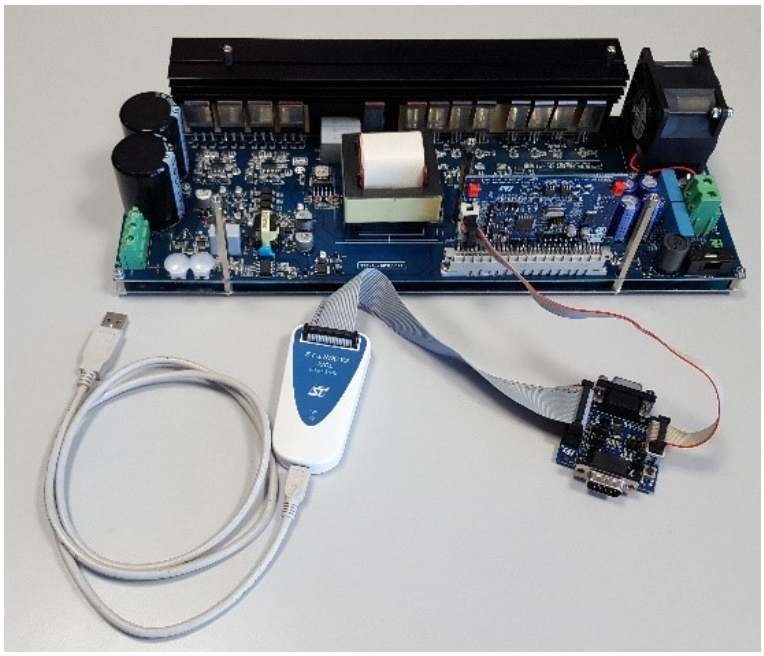
2. The on-board cooling fan should always be enabled; ensure that it is not disconnected as this may provoke system damage from overheating.
   - To perform efficiency measurements without fan consumption drawn from the output, remove jumper J4 and apply an external voltage (up to 48 V, 240 mA) to J5 connector
   - You can also disable the on-board fan control in the firmware or via UI, removing J4 before applying the external voltage
3. Ensure the power board is not powered
4. Connect the STEVAL-DPS334C1 digital control board to the power board through the 64 pin DSMPS connector (P1). The control board is already programmed and ready-to-use. To program a new control board, you should power it with an external 5 V supply through the control board J3 connector

Note: Do not connect the control board to the power board for this operation.

5. To update or debug the firmware, connect a 10-wire flat cable to J7 on the control board and J1 on the adapter board
6. Connect a standard programmer (such as ST-LINK or J-Link) through J3 standard 20-pin JTAG connector on the adapter board

Note: Do not apply any voltage to the power board for these operations.

Figure 21. **STEVAL-DPSLLCK1 connections**



Once the input power supply, the output load and the adapter board (optional for debugging, programming or to use the UI) are connected, the power supply is ready. As soon as the input voltage is above 100 V, the auxiliary power supply begins supplying the control board, drivers and signal conditioning circuitry. The presence of the input voltage can be recognized if the red LED (D64) on the power board and the green LED (D1) on the control board are on.

If the input voltage is within the proper range [375 V; 425 V], the system, after a default wait time of 2000 ms, starts the ramp-up procedure to avoid high current spikes. The converter state and any faults are signalled by the blue state LED and red fault LED.

3.5 Required and suggested test equipment

Required equipment:

- 1 x 3400 W programmable DC power source 400 V, 9 A. It is possible to use a lower current limit for educational purposes, reaching a lower power rating
- 1 x 60 V, 65 A electronic DC load. Instead of an electronic load, it is possible to use 3kW resistive load bench using liquid cooling system or fan cooling

Highly recommended equipment:

- 1 x digital oscilloscope to check voltage and current waveforms
- 1 or 2 x USB isolator (s) to be used when the UI interface and/or a debugger (not isolated) are connected

Recommended equipment:

- 1 x Rogowski coil or current Probe to sense resonant current
- 4 x high voltage differential and galvanically insulated voltage probes

Optional equipment:

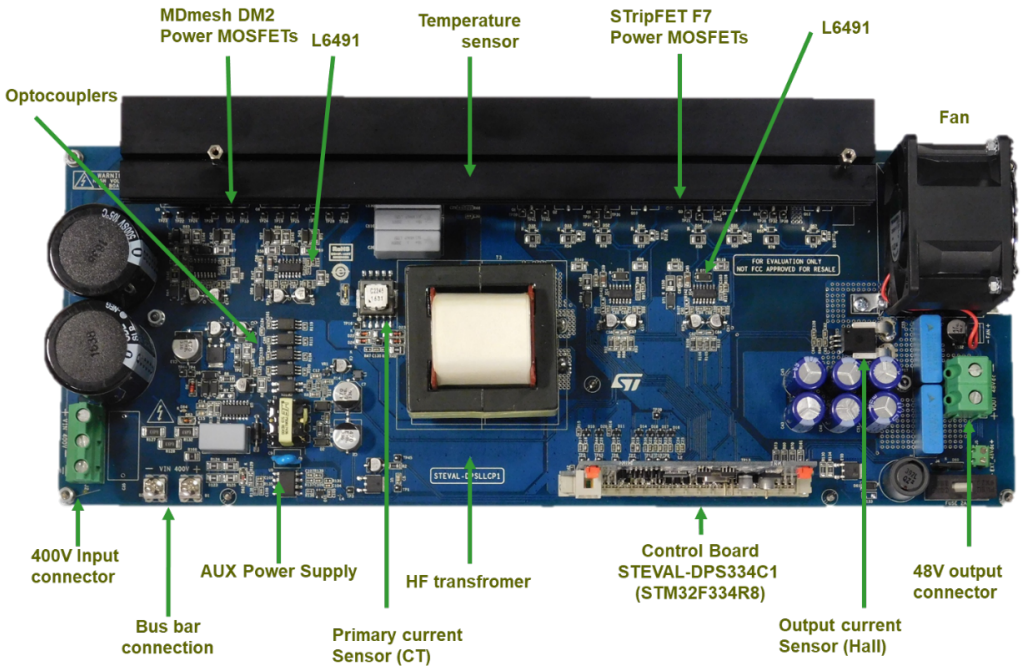
- 2 x digital multimeters (or a power meter) to measure input/output voltages
- 1 x ST-LINK programming tool (or another tool, like J-Link) to reprogram the board or for debugging
- 1 x adapter board to reprogram the board or for UI communication
- 1 x RS-232 dongle to connect the adapter board to a PC and use the UART interface
- STSW-DPSLLCK1 firmware package (last available version) for firmware customization
- IAR Embedded Workbench for ARM or another toolchain for firmware customization

4 Testing the converter

To modify the firmware or to enter debug mode, using a programming tool, it is possible to supply the power board with a 100 V_{DC} voltage to make the auxiliary power supply work. In this condition, the MCU detects an input undervoltage fault (PWM outputs automatically disabled) and can be safely reprogrammed using IAR Embedded Workbench for ARM ver.7.80 (or higher), or another programming toolchain.

4.1 Light load burst mode and SR

Figure 22. STEVAL-DPSLLCK1 kit overview



The STSW-DPSLLCK1 firmware package is set to operate in closed loop operation unless the operation mode is changed by the user, either at compile time or on-the-fly via UI. To test the closed loop operation and the way the board works, follow the procedure below.

- Connect the DC electronic load (max 65 A) to the 48 V output connector (J3) and a digital multimeter as close as possible to the connector
- Set the electronic load in CC mode (Continuous Current) and set 0 A (no load)
- Connect the galvanically insulated voltage probes to check the low side VGS voltages of SR MOSFETs to TP35-TP37 (Q9-Q10), and to TP36-TP38 (Q11-Q12) (one probe per couple of MOSFETs) and Rogowski coil or current probe around J1 strap to check the tank resonant current (e.g., if the probe has a sensitivity of 20 mV/A, set the channel scale at 100 mV/div to have 5 A/div)
- Connect the DC power supply to J2 400 V input connector together with a digital multimeter (as close as possible to the connector) and set 1 A as maximum current limitation
- Apply 400 V_{DC} with the DC power supply and check the regulation of output voltage at 48 V DC on electronic load or multimeter (no load operation): the control board blue LED lights up, while the red LED must be off. At no load, the output voltage may oscillate due to the hysteresis control (from 47 to 50 V, typically)

- Increase the DC-DC power supply current limitation to 9 A to make sure you can reach the maximum power of 3 kW.

Important: You should work with a lower limiting current for safety reasons if you are just learning to use the board.

- Set the output current at 1 A and check the burst mode operation (the resonant current is discontinuous, as shown in Figure 23
- Increase the output current up to 4 A: in this condition, the light load burst mode is automatically disabled (resonant current is a continuous waveform similar to a sinusoid), but SR gate signals are still not present, as shown in Figure 24

Figure 23. Resonant current and Vgs on SR MOSFETs @ Iout = 1 A: burst mode is ON, SR is OFF

The resonant current fluctuates (green trace) and the synchronous rectification (SR) gate signals are both low

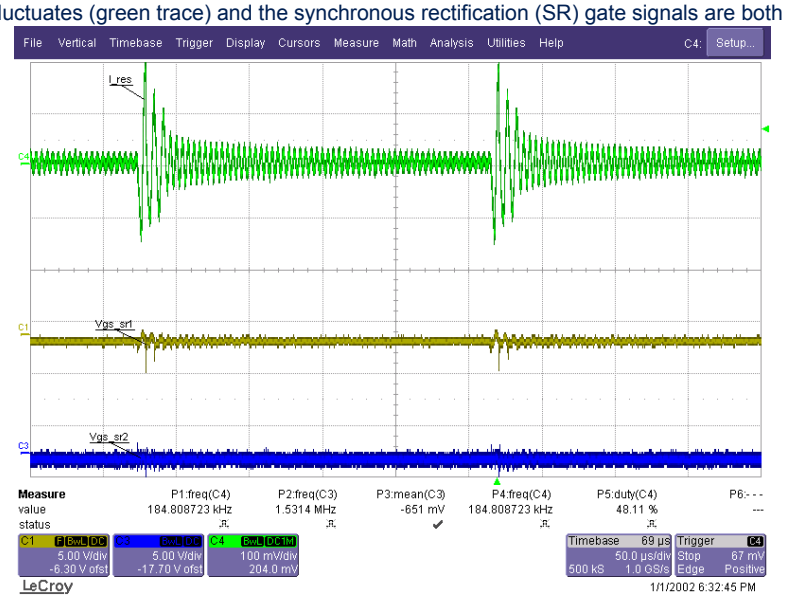
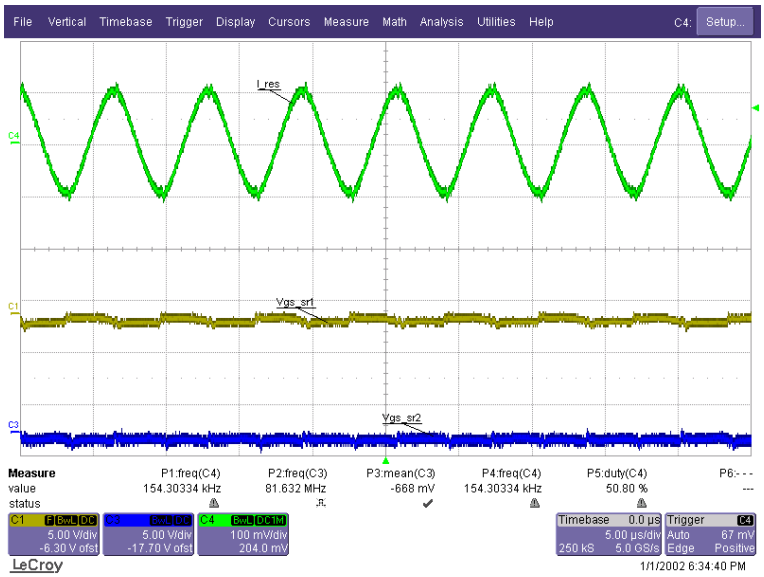
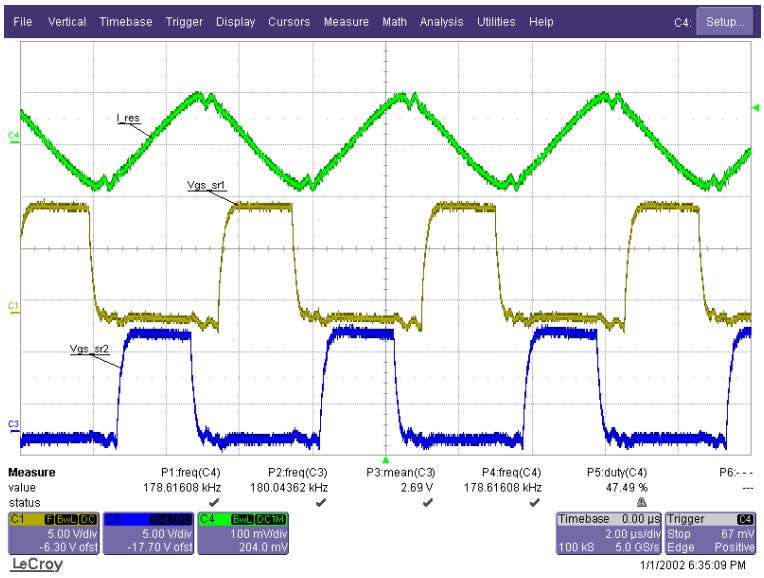


Figure 24. Resonant current and Vgs on SR MOSFETs @ Iout = 4 A: burst mode is OFF, SR is OFF



If the load is increased above 5-6 A, synchronous rectification is automatically enabled.

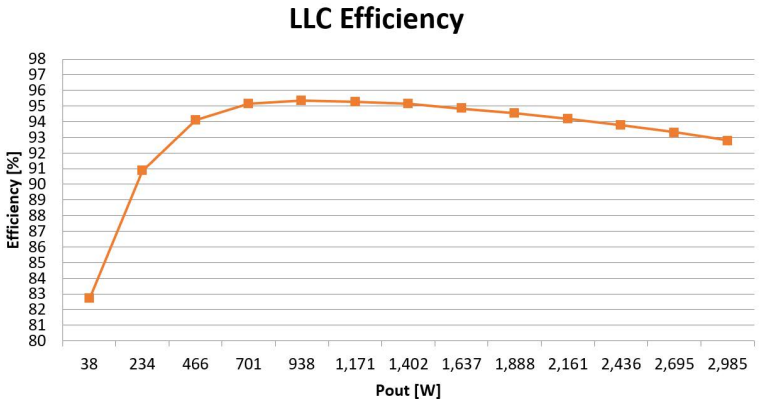
Figure 25. Resonant current and Vgs on SR MOSFETs @ Iout = 6 A: burst mode is OFF, SR is ON



The SR PWM signal timing diagram is shown in Figure 7. By default, adaptive synchronous rectification is enabled, thus falling delays are automatically changed.

The converter efficiency plot in closed loop operation and Adaptive SR enabled is shown in Figure 26. The efficiency curve is strongly dependent on design choices of the resonant tank: a wider input or output voltage range specification implies a lower efficiency. The board reaches 95% efficiency from 700 to 1400 W.

Figure 26. Efficiency results in closed loop operation and adaptive SR enabled



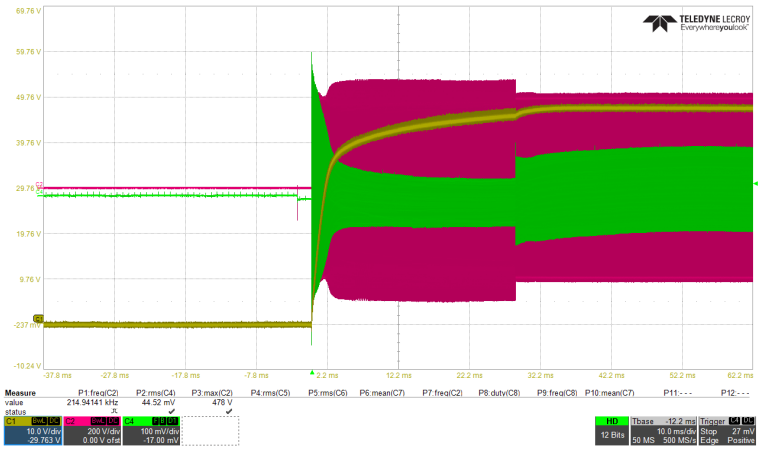
4.2 Power on and steady state

The following procedure describes how to check main waveforms at start-up while keeping closed the loop operation enabled.

- Remove the 400 V DC voltage and wait until the input and output voltage falls to zero
- Connect a high voltage insulated differential probe on the primary side to view the full bridge output voltage applied to the resonant tank (test points TP27 and TP28)
- Connect the Rogowski coil or another current probe around J1 strap to check the tank resonant current (e.g., if the probe has a sensitivity of 20 mV/A, set the corresponding channel scale to 100 mV/div to have 5 A/div)
- Supply the 400 V DC on the input connector and check the Vout ramp-up without load (0 A on the output) using a single trigger capture setting on the oscilloscope to obtain a result similar to the figure below.

Figure 27. Start-up waveforms at power-on

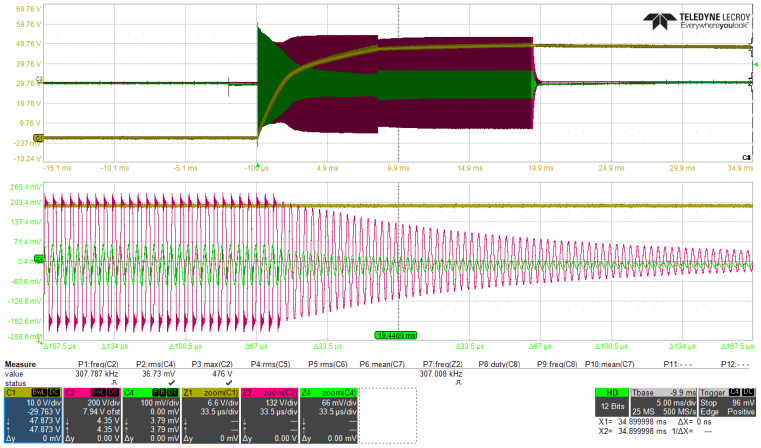
- Yellow trace: output voltage
- Green trace: tank resonant current
- Red trace: square voltage applied to the resonant tank



In the above figure, thanks to the start-up procedure, the output voltage (yellow trace) rises smoothly while the resonant tank current (green trace) is limited in amplitude. If the start-up procedure is disabled, the current spike would be much larger and the overcurrent protection would be triggered. When the output voltage reaches the minimum defined threshold (46 V), the control loop is closed and the PID regulator becomes effective (a small glitch on current and voltage can be seen when this occurs).

If no load is applied, after few PWM cycles, burst mode is triggered and a natural oscillation is present, even if output voltage continues to be controlled, as shown in the figure below.

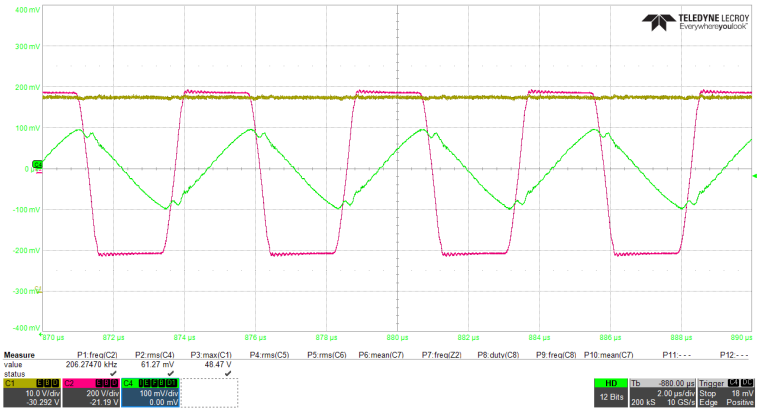
Figure 28. Voltages and resonant current when, after start-up procedure, burst mode starts (no load)



If the output load is increased, the light load is disabled, as shown in the following figure.

Figure 29. Steady state when a 4.5A load is applied

- Yellow trace: output voltage
- Green trace: tank resonant current
- Red trace: square voltage applied to the resonant tank



4.3 Light load burst mode and output voltage burst mode

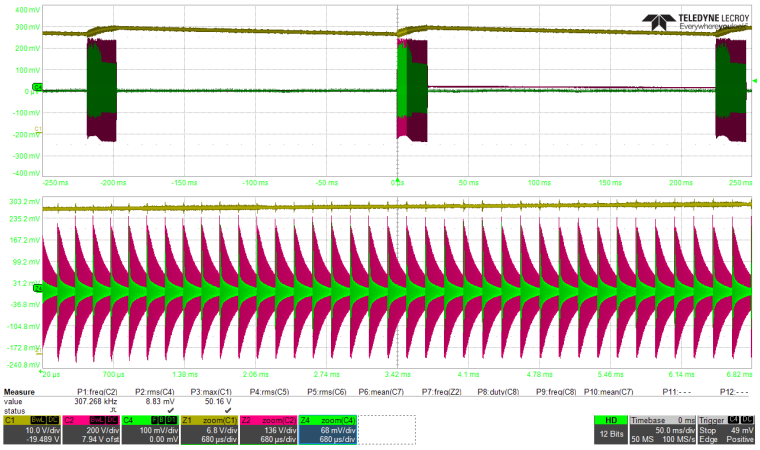
The STSW-DPSLLCK1 firmware package implements two types of burst mode: the first one based on output voltage measurement and the second one based on light load.

The first burst mode ensures the output voltage does not exceed an upper threshold limit without going into overvoltage protection. It can be triggered when no load is applied and depends on resonant tank minimum gain at maximum switching frequency in closed loop (it can vary a little according to transformer tolerances).

The second burst mode type is based on light load. When it is enabled, according to the output current measurement, some PWM pulses are skipped to reduce conduction losses due to circulating current on the primary side.

Figure 30. Triggered burst modes (first and second)

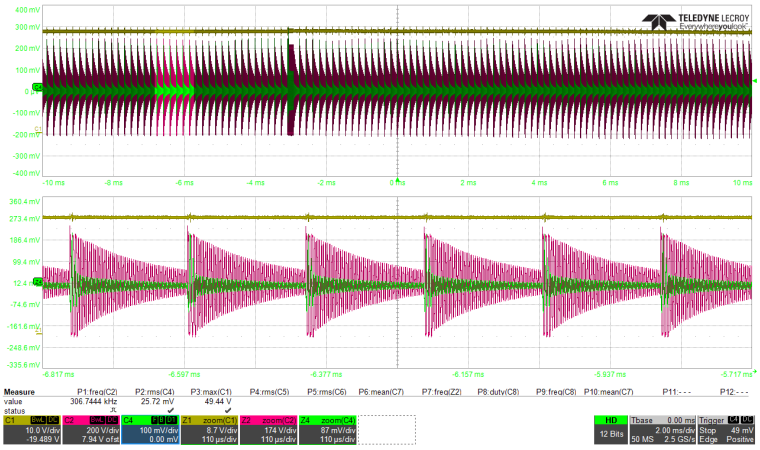
- Yellow trace: output voltage
- Green trace: tank resonant current
- Red trace: square voltage applied to the resonant tank



Increasing the load up to 0.7 A, only the light load burst mode is present, as shown below.

Figure 31. Light load burst mode

- Yellow trace: output voltage
- Green trace: tank resonant current
- Red trace: square voltage applied to the resonant tank



4.4 Synchronous rectification and open loop mode

The following tests can be performed using the UI interface to enter the related commands. You can use any application that can insert and read characters (e.g. HyperTerminal or PuTTY) and a USB to RS-232 serial adapter.

Note: If the serial communication dongle is not galvanically isolated, the use of a USB isolator is strongly recommended.

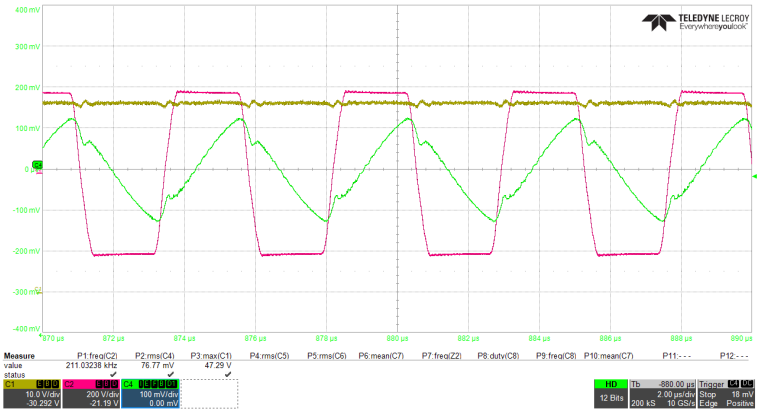
To check SR effects, perform the following steps:

- Turn the board on in closed loop mode and set output load at 10 A (at this load, SR is automatically enabled)
- Check the efficiency and switching frequency set by control loop in this condition (e.g., in Figure 32, the switching frequency is set to 211 kHz), the fan could be disabled to perform this measurement (by the command: fan off)
- Disable SR outputs (sr off)

When SR is disabled via user interface, the switching frequency changes and is automatically adjusted by the control loop to regulate the output voltage at 48 V. The frequency is lower than it was in the previous case. A lower efficiency can also be measured when the synchronous rectification is disabled (enable the fan again after taking measurements by the command: fan on).

Figure 32. Closed loop operation with Iout = 10 A, SR enabled

- Yellow trace: output voltage
- Green trace: tank resonant current
- Red trace: square voltage applied to the resonant tank



When SR is OFF, there are higher conduction losses in SR power MOSFETs as the current flows through the internal body diode. Moreover, to compensate the voltage drop of the internal body diode and to maintain the output voltage at the desired value, a reduction of switching frequency is necessary to increase the voltage gain of the resonant tank.

Looking at the figure above, we can see that the resonant current (green trace) lags the voltage applied to the resonant tank (red trace). This means that the impedance of the resonant tank has a behavior similar to an inductor (inductive region) and it is a normal condition for the resonant converter. Working in the inductive region allows the discharge of power MOSFET output capacitance during the dead time before the next turn-on, thereby achieving zero-voltage switching (ZVS) and the reduction of switching losses. The capacitive zone, in which the current leads the voltage, must be avoided (no ZVS, only ZCS), as, in this condition, the primary side power MOSFETs operate in hard switching and may be damaged. The EMI level is also higher in this condition.

To perform this test and check the converter is working in inductive region, pay attention to the polarity of voltage and current probes.

The following steps describe how to set open loop mode, manually change switching frequency and tune fixed delay SR.

- While the board is off, place differential isolated probes on low side SR1 power MOSFETs to view gate signal (TP35 to TP37) and corresponding Vds voltage (TP31 toT37)
- While the converter is working, disable PWM outputs (out off), set load at 10 A, disable SR (sr off), enable open loop mode (ol on), then enable output again (out on) and check output voltage and switching frequency: now Vout is lower than 48 V because the frequency is set (after start-up) at 250 kHz, so there is a lower voltage gain, as shown in Figure 33. In this condition, Vout undervoltage protection is automatically disabled, whereas overvoltage and burst mode are still disabled.
- Now decrease the open loop switching frequency (e.g., 25 kHz per step, so with the command: freq 225000) keeping the load constant, and check that Vout is increasing (Figure 34). This procedure must be performed while providing small frequency steps to avoid triggering output overvoltage protection, which is set at 56 V
- Set the switching frequency to 200 kHz (freq 200000, all commands entered so far are shown in Figure 35)
- SR outputs are still disabled; in this condition, disable adaptive SR (asr off) and enable SR outputs (SR on). In this way, the adaptive algorithm is disabled and SR PWM signals are generated according to the current switching frequency (200 kHz) and fixed delays (for falling delays, set to their maximum value by default: 600 ns). The result is shown in Figure 36: comparing it with Figure 10 a too early turn-off can be detected (diode conduction time is measured by oscilloscope to be 900 ns, this is also due to delays introduced by optocouplers and driving)
- SR fixed delays (delay rising 1, delay falling 1, delay rising 2 and delay falling 2) can be changed, even it a safe diode conduction time must always be considered to prevent board damage. By considering SR1 waveform, we can reduce falling delay 1 to 300 ns (df1 300). The result is shown in Figure 37, where we can see that diode conduction time has been reduced to 625 ns
- We can also enable adaptive SR algorithm (asr on) and verify that diode conduction time is automatically further reduced (Figure 38)

Figure 33. Open loop operation after start-up with Iout = 10 A, Fs = 250 kHz, Vout = 43 V

- Yellow trace: output voltage
- Green trace: tank resonant current
- Red trace: square voltage applied to the resonant tank

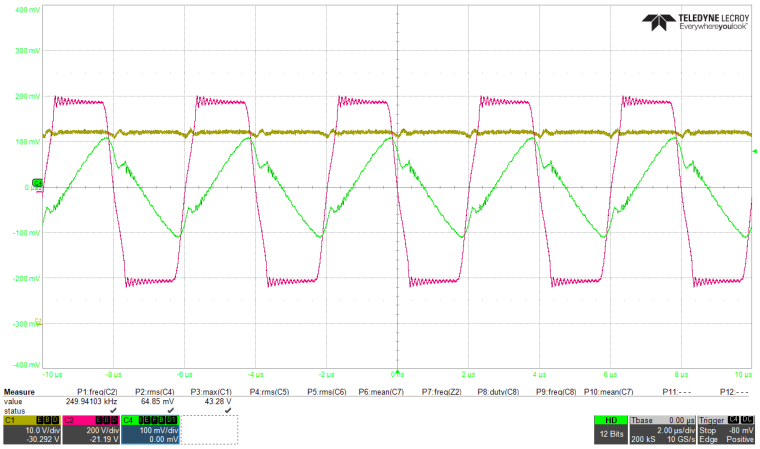


Figure 34. Open loop operation with Iout = 10 A, Fs = 225 kHz, Vout = 45 V

- Yellow trace: output voltage
- Green trace: tank resonant current
- Red trace: square voltage applied to the resonant tank

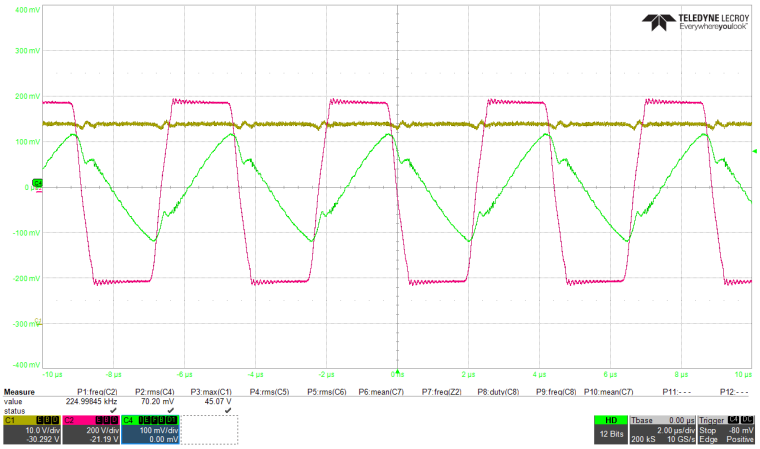


Figure 35. **UI command list for open loop test**

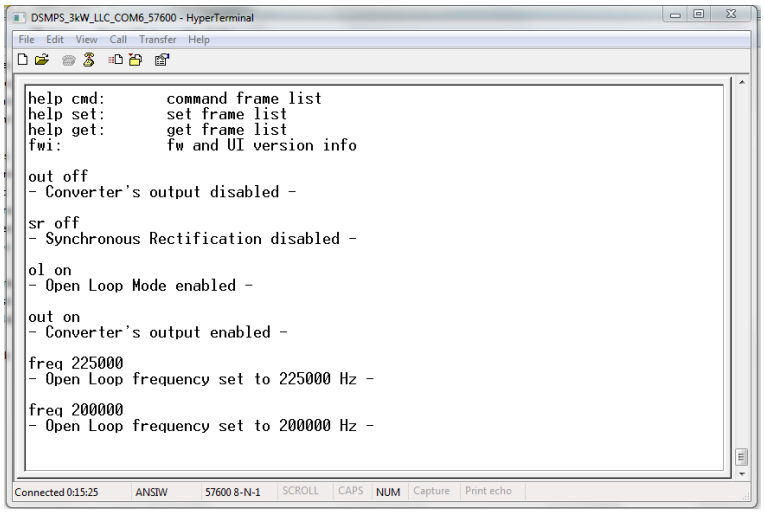


Figure 36. **SR fixed delays in open loop mode df1 = 600ns**

- yellow trace: Vds voltage SR low side 1
- green trace: tank resonant current
- red trace: Vgs voltage SR low side 1

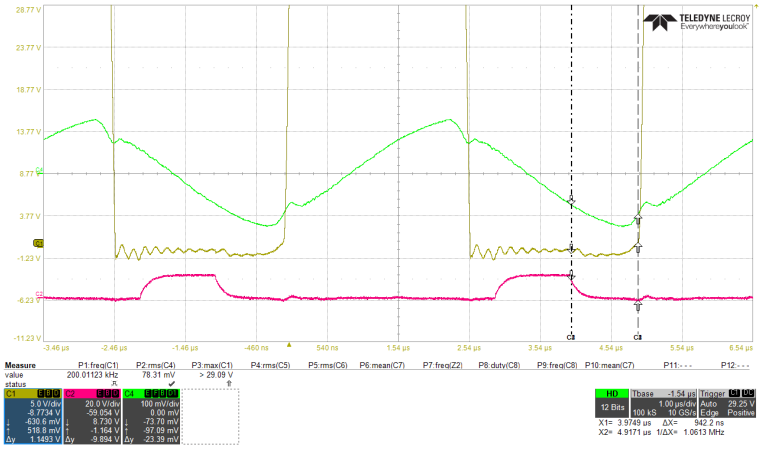


Figure 37. **SR fixed delays in open loop mode df1 = 300ns**

- yellow trace: Vds voltage SR low side 1
- green trace: tank resonant current
- red trace: Vgs voltage SR low side 1

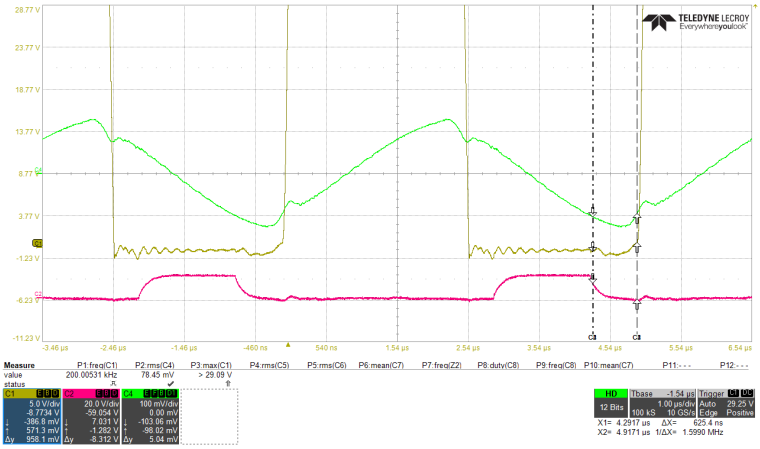
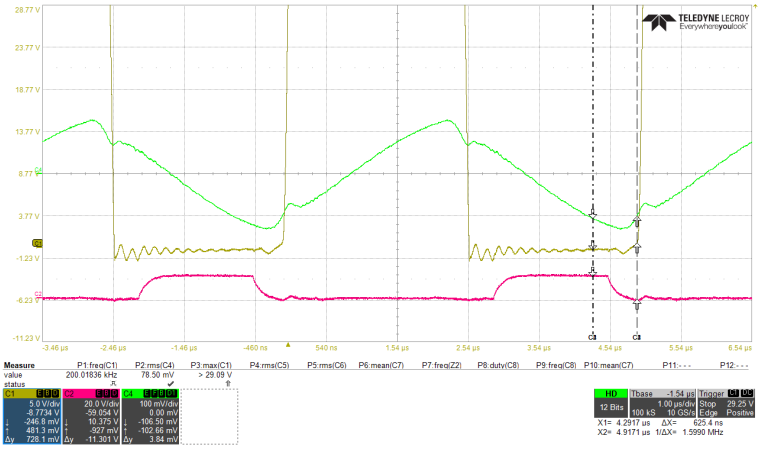


Figure 38. **Adaptive SR in open loop mode**

- yellow trace: Vds voltage SR low side 1
- green trace: tank resonant current
- red trace: Vgs voltage SR low side 1



4.5 Load step response

Tests about load step responses with closed loop and adaptive SR have been performed using a voltage probe (yellow trace in the figures below) and a current probe (blue trace in the figures below) connected to the load. Default PID values (Kp=3000, Ki= 1000, Kd=1000) have been used but they can be changed via UI, considering that transient behavior can change, output voltage may became unstable, and some protections may be triggered. The other commands (fan on/off, get measure, get params, etc.) can also be tested.

Figure 39. **Load step response: from 2 to 25 A**

- yellow trace: output voltage
- blue trace: output current
- green trace: tank resonant current
- red trace: voltage applied to resonant tank

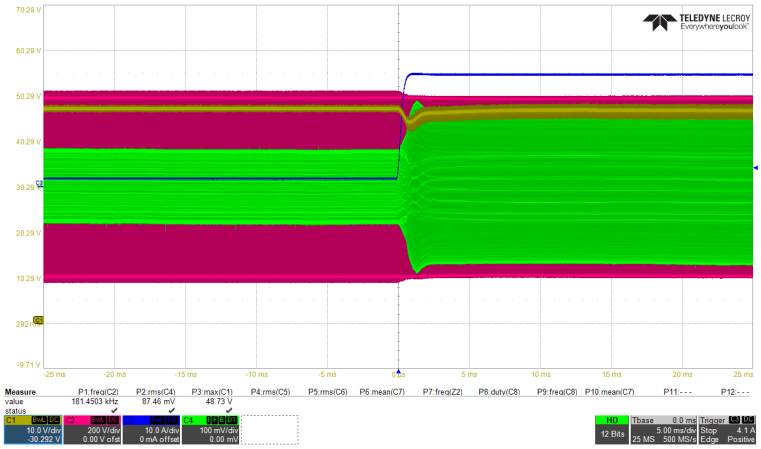
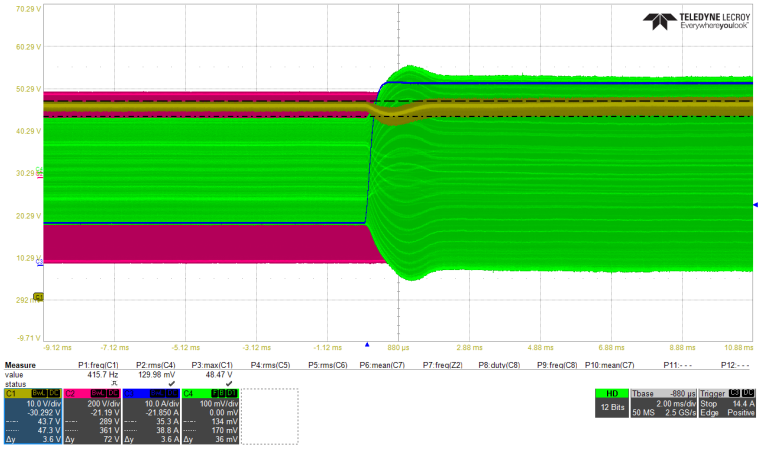


Figure 40. **Load step response: 10 to 40 A**

- yellow trace: output voltage
- blue trace: output current
- green trace: tank resonant current
- red trace: voltage applied to resonant tank



Revision history

Table 7. **Document revision history**

Date	Version	Changes
15-Oct-2020	1	Initial release.
12-Nov-2020	2	Minor text changes throughout the document.

Contents

List of tables

List of figures