Application note

Boost voltage mode on B-G474E-DPOW1 Discovery kit

Introduction

This document describes the contents of the Boost_VoltageMode_HW project, a software example running on the B-G474E-DPOW1 hardware. This low-cost and easy-to-use kit is convenient for quick evaluation and application development with STM32G4 Series microcontrollers, a family of devices designed for digital power conversion applications, combining high integration with performance.

The document illustrates how to drive two PSU DC-DC converters simultaneously (buck and boost), with voltage mode closed-loop regulation on the boost converter, by describing the steps required to execute the code and to check output signals, using the IAR Embedded Workbench® and STMicroelectronics STM32CubeIDE toolchains. The document describes the buck voltage mode usage with the X-CUBE-DPOWER STM32Cube Expansion Package. Finally, it lists the STM32G474xx capabilities exploited by this application, and how they can be used in larger scale systems.

1 General information

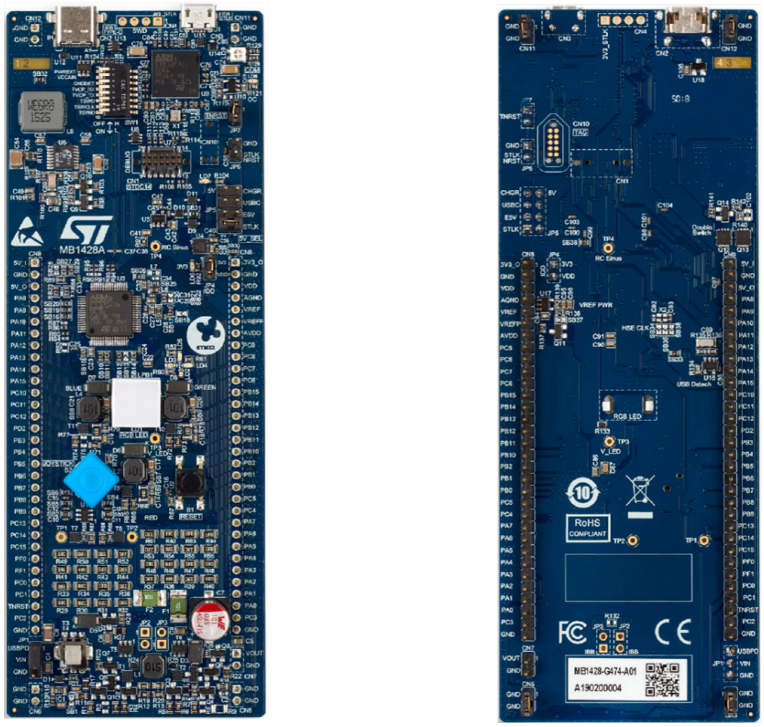
This document applies to the STM32G4 Series microcontrollers, based on Arm® cores.

Note: Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

2 Hardware board overview

The B-G474E-DPOW1 is a complete digital power starter kit controlled by the STM32G474RET6 microcontroller. The kit helps the user discover the features of digital power including LED dimming, buck/boost with variable load, power delivery (USB type-C™), and audio Class-D amplification.

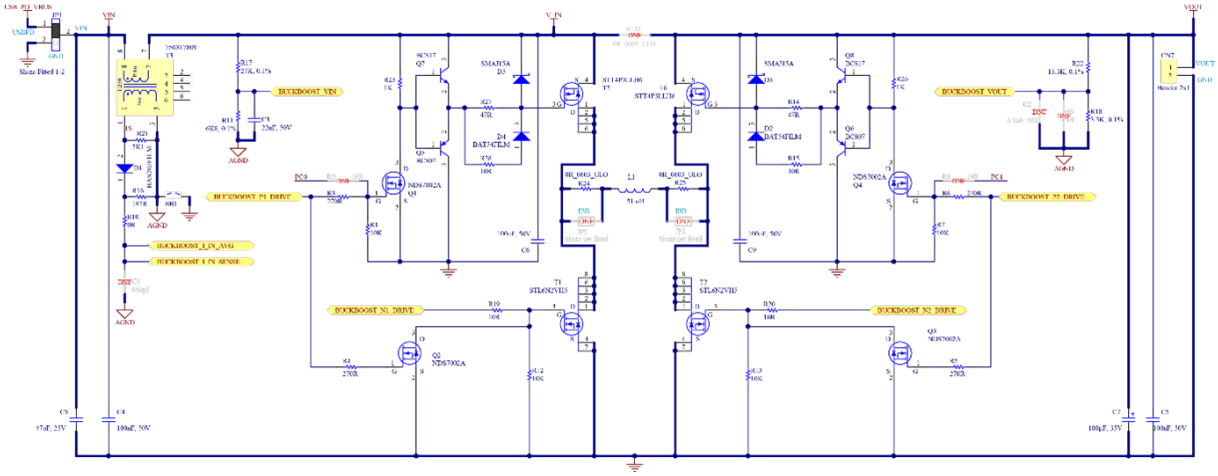
Figure 1. Discovery kit: top (left) and bottom (right) sides



2.1 Buck-boost converter

This kit embeds a buck-boost converter. There are both boost and buck switching FETs at each side of the converter's inductor. The extract from the schematic shown in Figure 2 identifies the relevant switches. The full schematic is available on www.st.com.

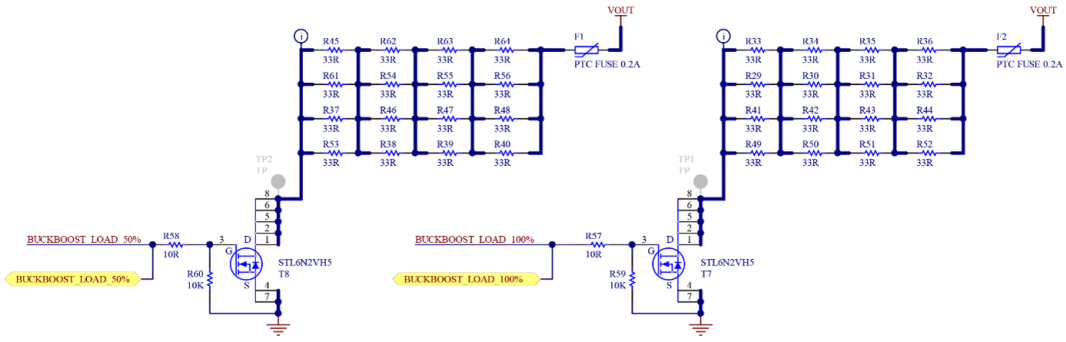
Figure 2. Buck-boost power stage schematic



2.2 Onboard load

This board includes two parallel load banks at the output of the converter, as shown in Figure 3.

Figure 3. Onboard load banks controlled via MOSFETs



The load banks are controlled by the MCU by toggling the PC14 and PC15 outputs, which drive, respectively, the BUCKBOOST_LOAD_50% and BUCKBOOST_LOAD_100% signals. The MOSFET switches the resistive load bank in and out of the circuit.

Therefore, when PC14 is high, load bank 1 is ON. When PC15 is high, load bank 2 is ON. The test pins TP1 and TP0 are also used to check the load activation status, although these are not populated. The load banks have the total resistance shown in Table 1.

Table 1. Onboard load steps

Load (%)	0% Load 1 OFF Load 2 OFF	50% Load 1 ON Load 2 OFF	100% Load 1 ON Load 2 ON
Load	∞	33 Ω	16.5 Ω
LED status	All OFF	Green	Green + Orange

The load status can also be observed via the green and orange LED toggling. Moreover, it can be modified with the use of the joystick on the board.

3 Application contents

This application implements a voltage mode closed-loop control on the boost converter, using the USB power delivery as power supply.

The USB-PD provides a 5 V supply, the board loads operate at a maximum voltage of 5 V, and the boost converter is applied by lowering the input voltage. Therefore, a lower input voltage is emulated by implementing buck conversion.

The buck converter lowers the output voltage to a value that the boost increases up to 5 V.

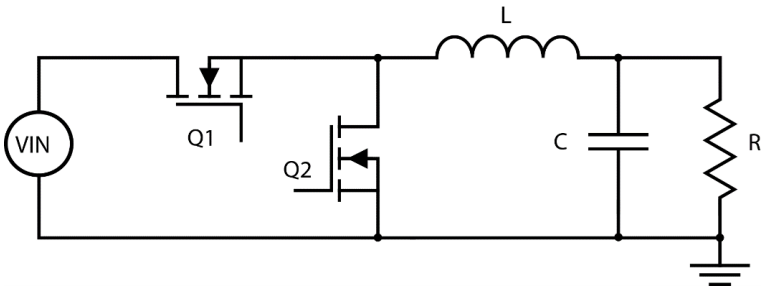
This application implements mechanisms that simultaneously lower and raise the output voltage, providing a 5 V regulated output voltage.

3.1 Buck converter open-loop operation

Operating principle

The kit contains a synchronous buck converter power stage (see Figure 4).

Figure 4. Simplified buck power stage schematic

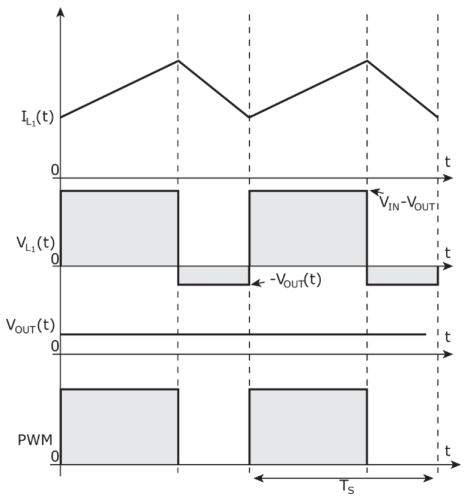


At the beginning of the switching period, the PWM of the top switch (Q1) is set high and the bottom switch (Q2) is set low. This turns on MOSFET Q1 and turns off MOSFET Q2. With the Q1 switch conducting, the current through the inductor L1 increases linearly. At the end of the high side duty cycle, the switch Q1 is turned off.

A dead time is inserted between high side and low side PWM for switches Q1 and Q2 to prevent shoot-through. After this dead time, the low side PWM for the switch Q2 goes high, turning on the switch Q2.

The inductor makes the current continue to flow through switch Q2. The current through the inductor decreases linearly. This switching action is described in Figure 5.

Figure 5. Buck converter operation waveforms



The C_{out} filters the AC component of this current, whose DC component is I_{out}.

As this is a step-down converter, the output voltage is always lower than or equal to the input voltage.

The steady-state output voltage depends upon the input voltage and the duty cycle, as shown in the equations below:

$$V_{OUT} = V_{IN} * D \quad (1)$$

$$D = PWM\_on\_time / PWM\_period \quad (2)$$

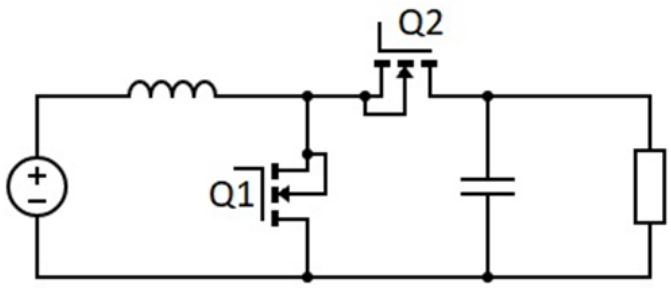
The buck is implemented in open loop with 50% duty cycle. This does not adequately regulate the output voltage upon load stepping, hence a voltage mode regulation is applied on the boost stage.

3.2 Boost converter closed-loop operation

Operating principle

The kit contains a synchronous boost converter power stage (see Figure 6).

Figure 6. Simplified boost power stage schematic



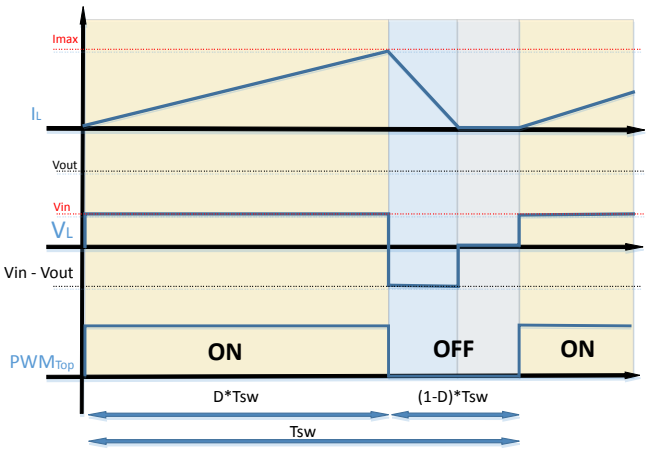
At the beginning of the switching period, the PWM of the bottom switch is set high and the top switch is set low. This turns MOSFET Q1 on, and MOSFET Q2 off.

The current through the inductor increases linearly. The output capacitor is already charged in steady-state and holds the output voltage on the load. At the end of the duty cycle, the switch Q1 is turned off.

A dead time is inserted between low side and high side PWM for switches Q1 and Q2 to prevent shoot-through. When this dead time has elapsed, the high side PWM for the switch Q2 goes high turning on the switch Q2.

The inductor current decreases linearly when it charges the output capacitor again (output voltage is higher than input), and current flows through switch Q2. This switching action is described in Figure 7.

Figure 7. Boost converter operation waveforms



The output filter capacitor filters the AC component, while the DC component is the output load current. The output voltage is always greater than or equal to the input voltage.

The steady-state output voltage is dependent upon the input voltage and the duty cycle:

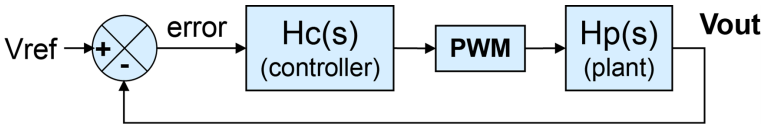
$$V_{OUT} = \frac{V_{IN}}{1 - D} \quad (3)$$

The voltage mode control loop is implemented to ensure reference tracking in the event of disturbances (such as load-stepping or supply variation). A changing voltage provided by the buck converter is associated to a line variation.

3.2.1 Voltage mode control loop

A typical power supply control loop is schematized in Figure 8.

Figure 8. Generic schematic of a PSU control loop



The classic procedure for designing the controller is to model the system in the frequency domain, then select the best controller type to compensate for the system behavior, obtaining adequate transient and stationary regulation.

When a mathematical model is known for the system, the compensator is calculated from that model.

The compensator can also be calculated empirically by measuring the system response, and designing a controller that compensates for the measured behavior.

Note: The control mode (for example, current or voltage mode) affects the system behavior.

Implementing voltage mode on a boost topology is not very common (regulation is poor compared to current mode, as shown in Section 5 Results of measurements). The buck plant is running at the same time, making it suitable for the plant response measurement method for identifying the compensator, as the mathematical models are not common for this operation.

The process of measuring the plant frequency response and placing the compensator poles and zeroes is not discussed here. Several workshops are provided by Biricha Digital to fine-tune controllers with this method.

The resulting compensator is a type 3 filter (three poles, three zeroes IIR filter).

Figure 9. Schematic of the implemented voltage mode control loop

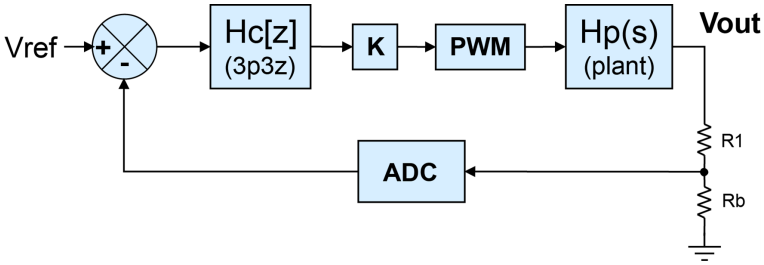


Figure 9 shows the power stage containing the switching MOSFETs, output filter inductor, and output filter capacitor within the block Hp(s). The controller, designed as a 3p3z filter, is the block Hc(z). This controller transfer function is expressed in the Z domain as it is implemented by digital means.

Digital voltage mode PSU control implies:

- the presence of the ADC sampling
- a scaling factor K
- the PWM generator

In Figure 9 the output voltage of the power stage is returned and compared with a reference, V_{REF}. The controller output is the duty cycle, which is used as an input to the power stage and modulates the MOSFET switches.

3.3 Load regulation

The user controls the load switching using the joystick supplied with the kit as follows:

- Left: decrease load by 50% (from 100 to 0%)
- Right: increase load by 50% (from 0 to 100%)
- Up: enable automatic load toggling (50 to 100%), useful for transient measurements
- Down: manual load selection

The status of the load bank during the transient is indicated by the LEDs:

- green LED: bank 1 is enabled
- orange LED: bank 2 is enabled

3.4 Software implementation

This section describes the peripheral configurations required to drive the switches of the open loop buck converter, and to perform the voltage mode control loop over the boost converter.

Note: The peripheral configuration required for the load regulation is a simple usage of the SysTick handler and GPIO-generated interrupts.

3.4.1 Peripheral configuration: open loop buck

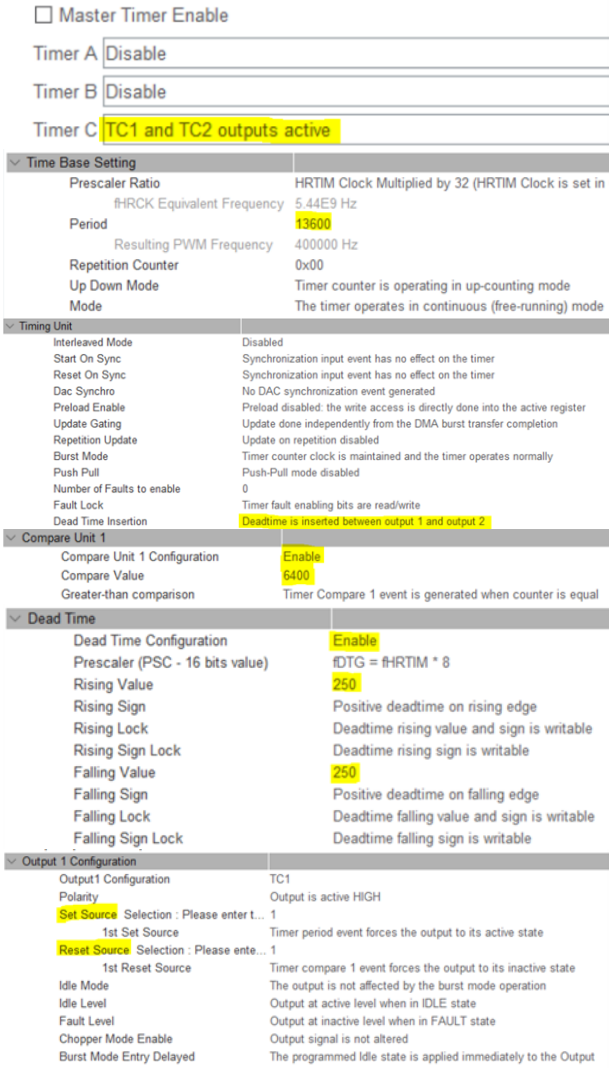
The MCU resources needed to drive the buck switches are two opposite high-resolution PWM outputs with a constant duty cycle. It is possible to use one of the channels of the HRTIM1 IP, available on STM32G474.

The buck switches are driven by channel C of the HRTIM1, which outputs two opposite signals at 400 kHz frequency, with a 370 ns deadtime on both edges.

It is necessary to program only one comparator corresponding to half of the period, as the converter operates with a constant duty cycle of 50%.

This results in the following STM32CubeMX configuration within the HRTIM1 section, which can be viewed in the `Boost_VoltageMode_HW.ioc` file.

Figure 10. **STM32CubeMX HRTIM1 Channel C configuration**



Once the timer is configured via STM32CubeMX, the user must activate the timer channel and outputs, from main.c before entering the infinite loop.

Note: The output 2 setting is not represented. When the dead time is enabled, it is automatically controlled by the settings of output 1.

3.4.2 Peripheral configuration: voltage mode closed loop boost

The MCU resources needed to run a voltage mode control loop on the boost converter are:

- Two complementary high-resolution PWM outputs with programmable duty cycle and deadtime insertion using one of the channels of the IP HRTIM1, available on STM32G474.
- An ADC channel to sample the output voltage, triggered synchronously with the PWMs. This is performed via the ADC1 and the IP interconnect in STM32G474.
- A digital filter processing engine, which allows the CPU to be unloaded when calculating the compensator. This is provided by the FMAC IP in STM32G474.
- A DMA channel to transfer the measured output voltage from the ADC to the FMAC at the end of each conversion.

HRTIM1 STM32CubeMX configuration

The boost switches are driven by Channel D of the HRTIM1, which outputs two opposite signals at 200 kHz, with a 370 ns deadtime insertion on both edges.

Two comparators must be programmed, one modifying every period with the value calculated by the compensator, and another representing the maximum duty cycle value, to limit the maximum output current.

The HRTIM1 must be configured to activate ADC conversion on the boost switching frequency, assigning the D channel period.

An additional comparator can be programmed to initiate the ADC conversion later in the period, thus improving the system stability and potential noise due to switching currents. For more information see AN5496, available on www.st.com.

STM32CubeMX configuration within the HRTIM1 section is available in the `Boost_VoltageMode_HW.ioc` file.

Figure 11. STM32CubeMX HRTIM1 Channel D configuration

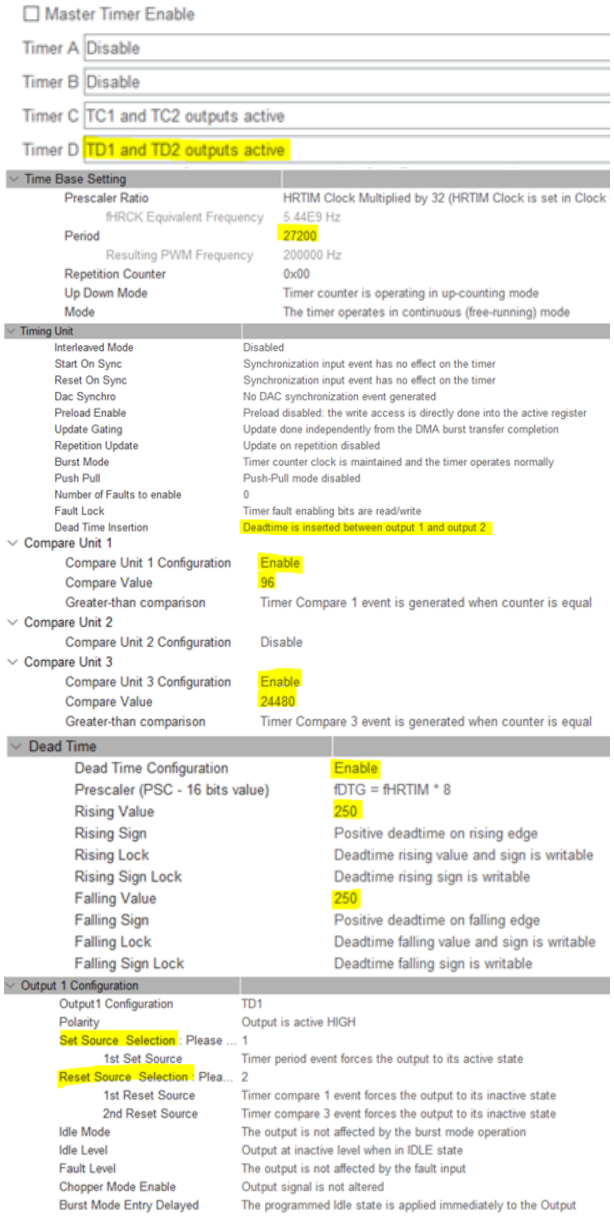
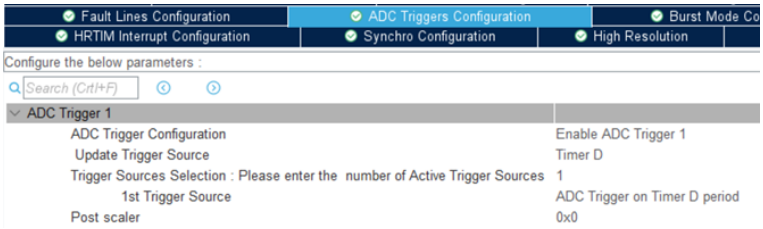


Figure 12. **STM32CubeMX HRTIM1 ADC triggers configuration**



The user needs to activate the timer channel and outputs in main.c before entering the infinite loop.

DMA1 STM32CubeMX configuration

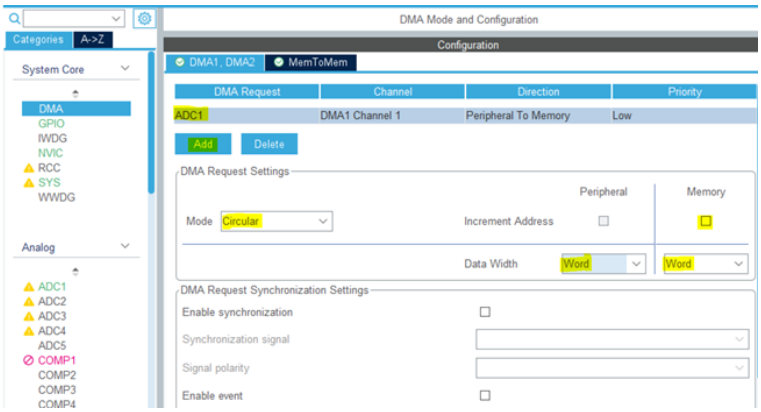
The DMA is used to copy the content of the ADC result register into the FMAC for the execution of the controller. By default, there is no DMA requests configured.

Channel 1 of DMA1 is used on the ADC1 conversion trigger. It transfers one data word each time, without incrementing the target address.

This transfer is performed from DMA1 to the FMAC data write register.

STM32CubeMX configuration within the HRTIM1 sections is available in the `Boost_VoltageMode_HW.ioc` file.

Figure 13. **STM32CubeMX DMA configuration**



ADC1 STM32CubeMX configuration

The ADC needs to be configured to sample the output voltage downsized by the resistor divisor. This lowered voltage is connected to PA3, configured to ADC channel 4 (`BUCBKOOST_VOUT` signal in the board schematic).

The ADC has a resolution of up to 12 bits. These values are stored in a 16-bit register with left or right alignment. The measured values are stored as left aligned. This means that the upper 12 bits (plus 1 sign bit) of the 16-bit register are used. This alignment must be considered for FMAC initialization.

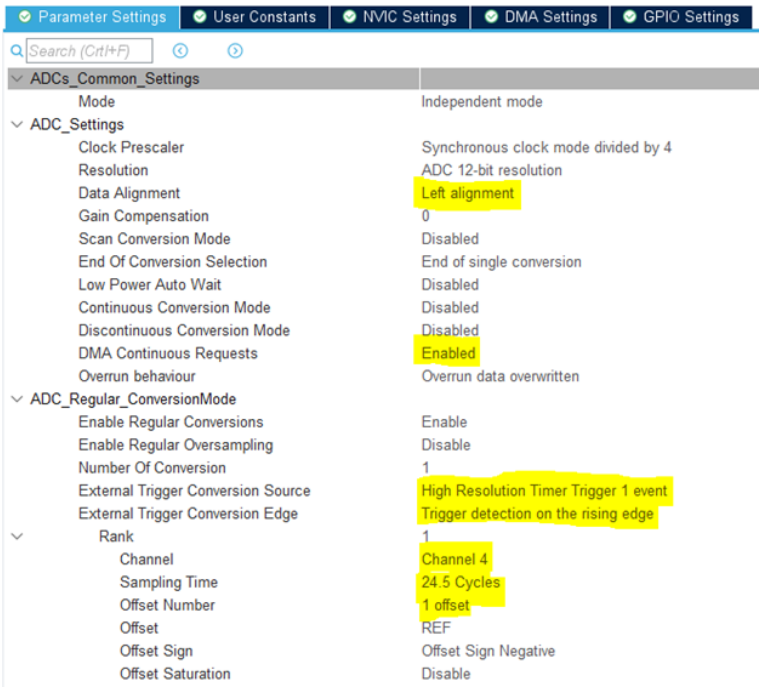
DMA requests are generated after each end of conversion. The conversions are triggered by an external conversion source given by the ***high resolution trigger 1*** event. This event is detected on the rising edge.

Only one ADC conversion is required. This conversion can be configured by expanding the ***rank*** subcategory.

Set channel to ***channel 4*** and configure the offset number to ***1 offset***. The ADC offset function subtracts an offset from the ADC value before it is stored in the result log. The formula $V_{ERR} = V_{REF} - V_{OUT}$ is used to perform the error calculation.

Set the offset value to ***REF***. To insert the REF value, click on the gear symbol and change the tick to ***no check***.

Figure 14. STM32CubeMX configuration of ADC1 conversion



Note: The high impedance of the potential divider on the board causes a drop of the ADC voltage when the sample and hold capacitor is connected. It takes around 500 ns for the voltage on the ADC pin to recover.

To maintain a consistent REF voltage reference value, take the sample after the capacitor has been charged. The charging rate of the capacitor depends upon the load.

The sampling time has been modified to 24.5 cycles (720 ns as the ADC clock prescaler is configured to four by default).

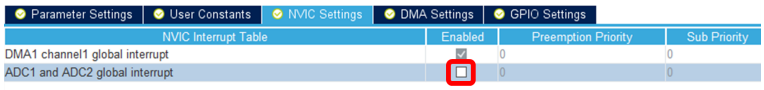
Note: See the Analog-to-digital conversion characteristics section of the product datasheet to adapt the sampling time to the actual impedance on the user board.

Figure 15. Scope plot of the ADC pin

In timing-critical applications, the performance of the ADC is optimized with a lower impedance at the potential divider.

The ADC1 interrupt is not used as the controller is automatically computed via the FMAC. The only use of the CPU is to update the PWM duty cycle with the value computed from FMAC.

Figure 16. **STM32CubeMX configuration of ADC1 interrupts**



After configuring the ADC via STM32CubeMX, the user must initiate the automatic calibration of the ADC in main.c before entering the infinite loop.

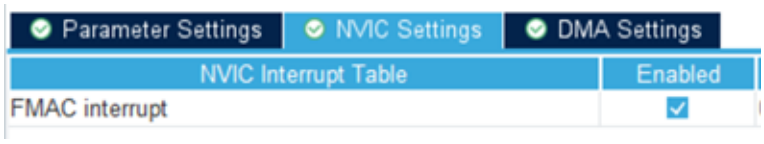
FMAC STM32CubeMX configuration

The FMAC is configured via the HAL library in the main.c.

However, the IRQ must be enabled, as it is used to update the PWM duty cycle each time a new value is calculated.

It is also possible to activate the IRQ directly from the FMAC section in the computing devices category of STM32CubeMX.

Figure 17. **STM32CubeMX FMAC IRQ configuration**



The FMAC is also initialized before entering the infinite loop, with the compensator coefficients declared in main.h.

After such configuration, the DMA destination for channel 1 requests can be associated to FMAC, and the ADC conversions can be started. This is performed in main.c before entering the infinite loop.

3.4.3 Program flow

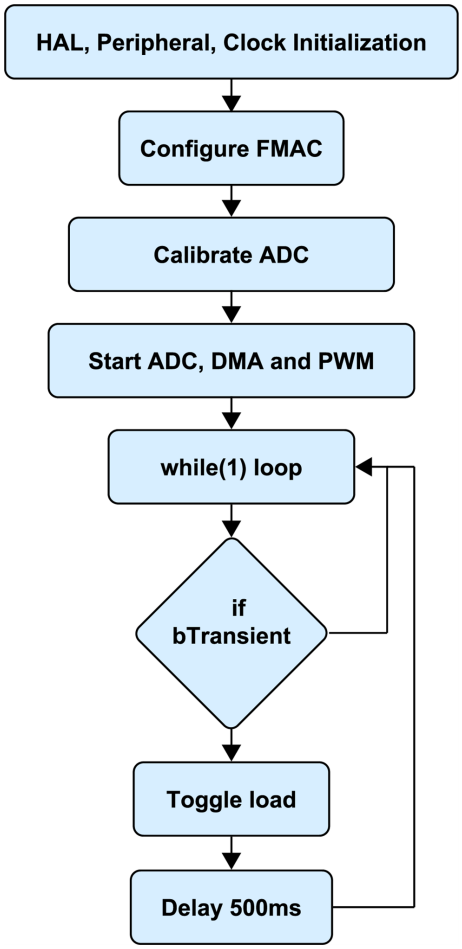
The main.c file contains most of the code for configuring and initializing devices. STM32CubeMX tool generates part of the code (see STM32CubeMX documentation on www.st.com).

Any additional code must be inserted between ***"user code starts here"*** and ***"user code ends here"*** comments inside the code files.

Outside of these comments, the code is automatically generated by the STM32CubeMX tool. If any changes are made to the project configuration within the tool, and the code is re-generated by clicking ***"generate code"***, the additional changes made outside these comments are lost.

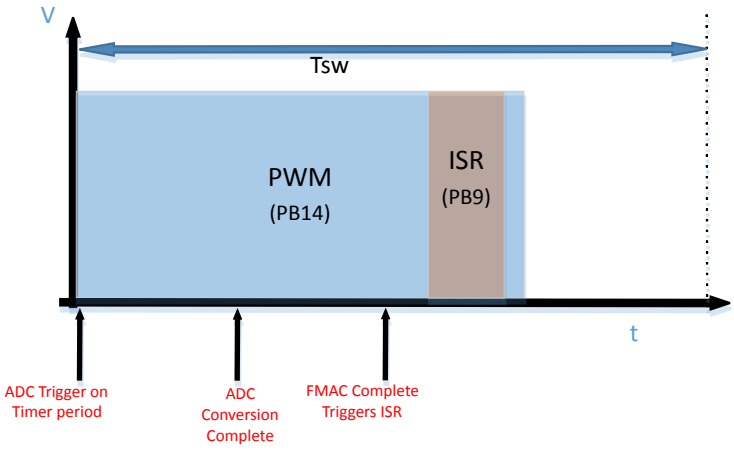
The structure of the main function within main.c is represented in Figure 18. This function contains calls to all initialization functions to configure the MCU peripherals. Next, the controller for the boost converter is set up and initialized with the calculated coefficients. Finally, the ADC is started and begins sampling the output voltage. The PWM outputs are enabled to drive the buck and boost switches.

Figure 18. Function flow of main() within main.c



The ADC module sampling is triggered by the HRTIM module (timer D period event). Once sampling and conversion are completed, the ADC activates DMA to copy the result directly from the ADC result log to the FMAC input register. This process is illustrated in Figure 19.

Figure 19. **ADC, DMA and FMAC timing diagram**



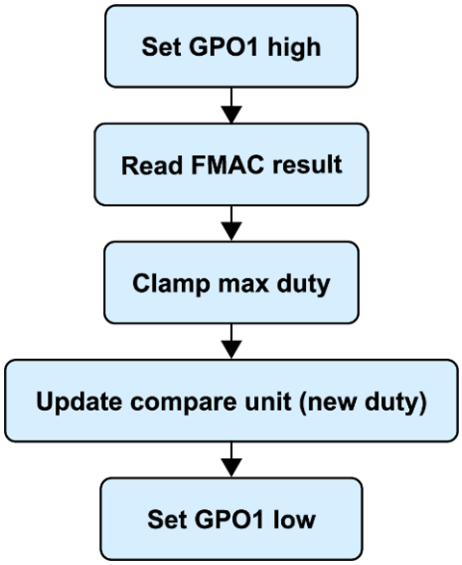
Interrupt service routine

After FMAC has completed running the 3p3z controller, it triggers an interrupt. This causes a jump of the MCU. The FMAC ISR is a separate function found towards the end of the `stm32g4xx_it.c` file included in this project. Its function name is `FMAC_IRQHandler (void)`.

The purpose of this FMAC ISR is to perform limit checking on the 3p3z controller output performed using the FMAC. The controller output is the new duty cycle value. This value is written in the register of the comparison unit 1 of the HRTIM.

The comparison unit has a maximum value, so the logic shown in Figure 20 is implemented within the user code section of the FMAC ISR.

Figure 20. **Flow of the FMAC ISR**



3p3z controller coefficients

The coefficients of the 3p3z controller are defined in `main.h`, at the top of the `main.c` file, by right-clicking on `main.h` and selecting `Open main.h` from the menu.

This header file contains several definitions for the pin names automatically generated by STM32CubeMX.

Further down this file, there is a section `/ * USER CODE BEGIN Private defines * /` where the definitions for the FMAC configuration start. The controller coefficients are defined under the FMAC configuration parameters. These coefficients (B0, B1, B2, B3, A1, A2, A3) are given in fixed-point hexadecimal form.

3.5 Software implementation

This section describes the peripheral configurations required to drive the switches of the open loop buck converter, and to perform the voltage mode control loop over the boost converter.

Note: The peripheral configuration required for the load regulation is a simple usage of the SysTick handler and GPIO-generated interrupts.

3.5.1 Peripheral configuration: open loop buck

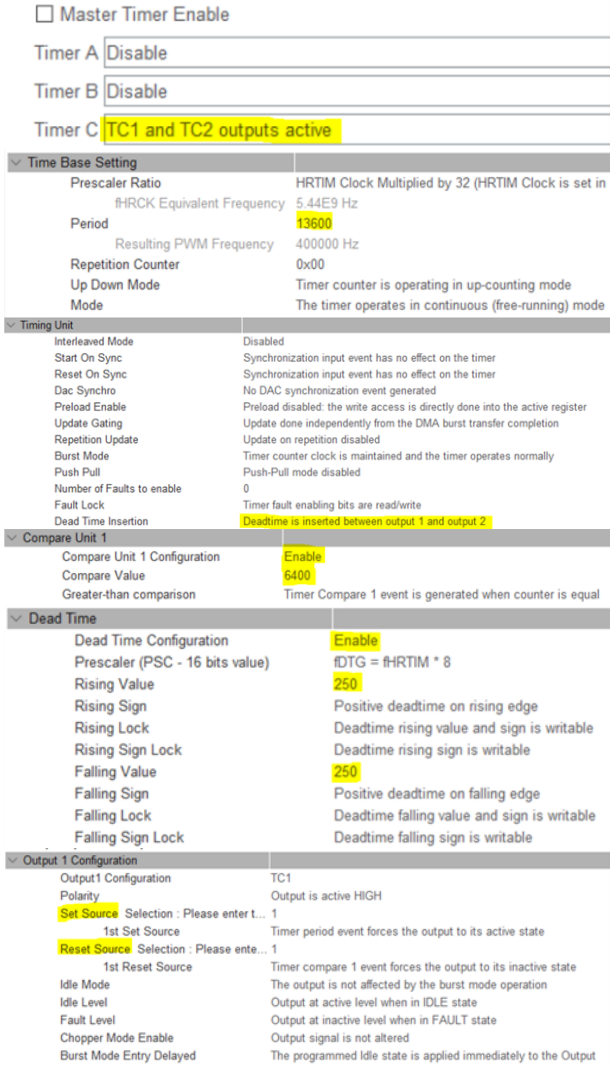
The MCU resources needed to drive the buck switches are two opposite high-resolution PWM outputs with a constant duty cycle. It is possible to use one of the channels of the HRTIM1 IP, available on STM32G474.

The buck switches are driven by channel C of the HRTIM1, which outputs two opposite signals at 400 kHz frequency, with a 370 ns deadtime on both edges.

It is necessary to program only one comparator corresponding to half of the period, as the converter operates with a constant duty cycle of 50%.

This results in the following STM32CubeMX configuration within the HRTIM1 section, which can be viewed in the `Boost_VoltageMode_HW.ioc` file.

Figure 21. STM32CubeMX HRTIM1 Channel C configuration



Once the timer is configured via STM32CubeMX, the user must activate the timer channel and outputs, from main.c before entering the infinite loop.

Note: The output 2 setting is not represented. When the dead time is enabled, it is automatically controlled by the settings of output 1.

3.5.2 Peripheral configuration: voltage mode closed loop boost

The MCU resources needed to run a voltage mode control loop on the boost converter are:

- Two complementary high-resolution PWM outputs with programmable duty cycle and deadtime insertion using one of the channels of the IP HRTIM1, available on STM32G474.
- An ADC channel to sample the output voltage, triggered synchronously with the PWMs. This is performed via the ADC1 and the IP interconnect in STM32G474.
- A digital filter processing engine, which allows the CPU to be unloaded when calculating the compensator. This is provided by the FMAC IP in STM32G474.
- A DMA channel to transfer the measured output voltage from the ADC to the FMAC at the end of each conversion.

HRTIM1 STM32CubeMX configuration

The boost switches are driven by Channel D of the HRTIM1, which outputs two opposite signals at 200 kHz, with a 370 ns deadtime insertion on both edges.

Two comparators must be programmed, one modifying every period with the value calculated by the compensator, and another representing the maximum duty cycle value, to limit the maximum output current.

The HRTIM1 must be configured to activate ADC conversion on the boost switching frequency, assigning the D channel period.

An additional comparator can be programmed to initiate the ADC conversion later in the period, thus improving the system stability and potential noise due to switching currents. For more information see AN5496, available on www.st.com.

STM32CubeMX configuration within the HRTIM1 section is available in the `Boost_VoltageMode_HW.ioc` file.

Figure 22. **STM32CubeMX HRTIM1 Channel D configuration**

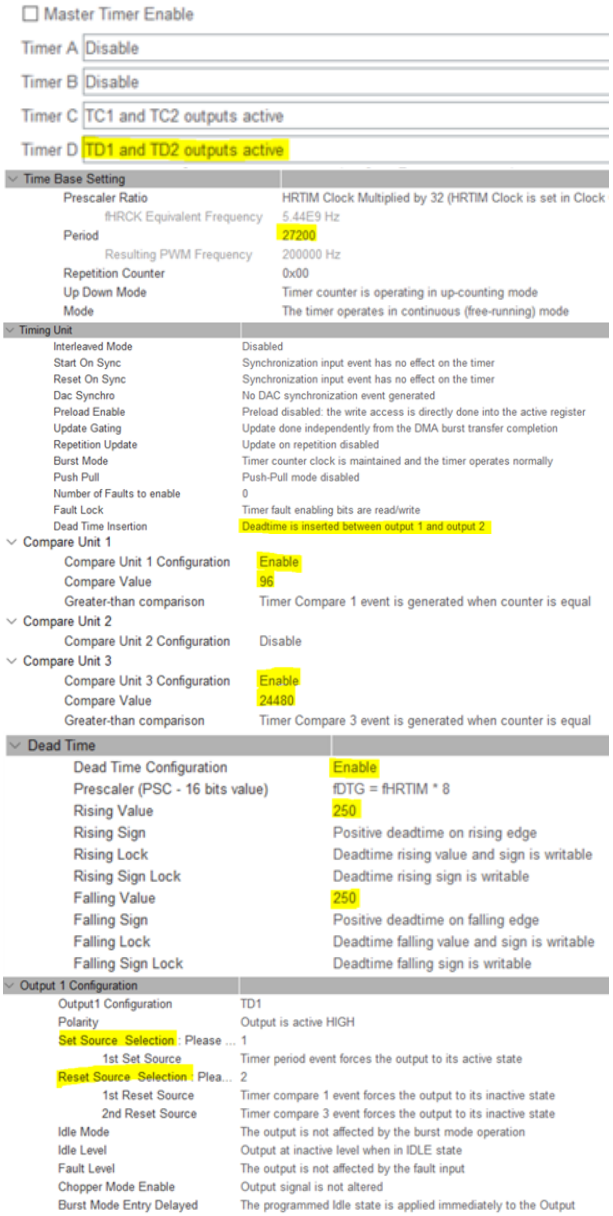
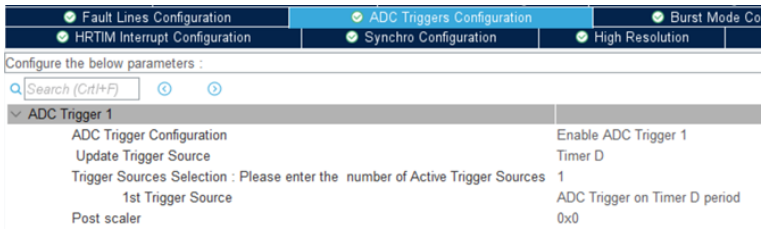


Figure 23. **STM32CubeMX HRTIM1 ADC triggers configuration**



The user needs to activate the timer channel and outputs in main.c before entering the infinite loop.

DMA1 STM32CubeMX configuration

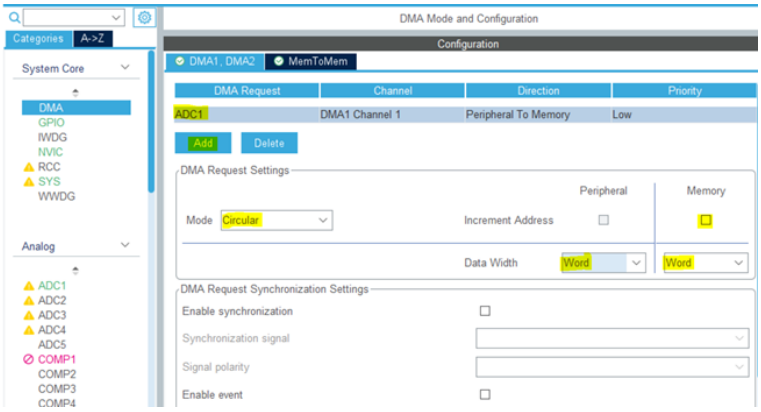
The DMA is used to copy the content of the ADC result register into the FMAC for the execution of the controller. By default, there is no DMA requests configured.

Channel 1 of DMA1 is used on the ADC1 conversion trigger. It transfers one data word each time, without incrementing the target address.

This transfer is performed from DMA1 to the FMAC data write register.

STM32CubeMX configuration within the HRTIM1 sections is available in the `Boost_VoltageMode_HW.ioc` file.

Figure 24. STM32CubeMX DMA configuration



ADC1 STM32CubeMX configuration

The ADC needs to be configured to sample the output voltage downsized by the resistor divisor. This lowered voltage is connected to PA3, configured to ADC channel 4 (`BUCBKOOST_VOUT` signal in the board schematic).

The ADC has a resolution of up to 12 bits. These values are stored in a 16-bit register with left or right alignment. The measured values are stored as left aligned. This means that the upper 12 bits (plus 1 sign bit) of the 16-bit register are used. This alignment must be considered for FMAC initialization.

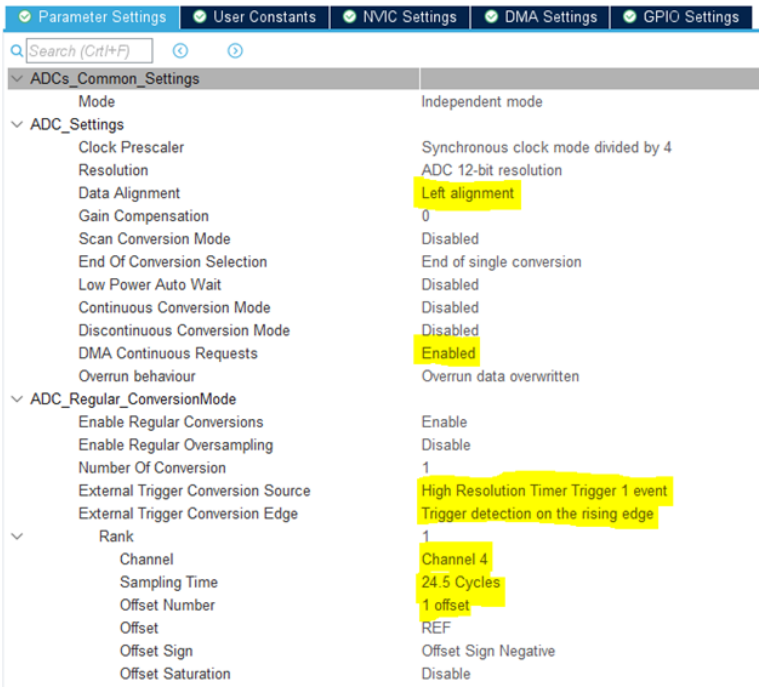
DMA requests are generated after each end of conversion. The conversions are triggered by an external conversion source given by the ***high resolution trigger 1*** event. This event is detected on the rising edge.

Only one ADC conversion is required. This conversion can be configured by expanding the ***rank*** subcategory.

Set channel to ***channel 4*** and configure the offset number to ***1 offset***. The ADC offset function subtracts an offset from the ADC value before it is stored in the result log. The formula $V_{ERR} = V_{REF} - V_{OUT}$ is used to perform the error calculation.

Set the offset value to ***REF***. To insert the REF value, click on the gear symbol and change the tick to ***no check***.

Figure 25. STM32CubeMX configuration of ADC1 conversion



Note: The high impedance of the potential divider on the board causes a drop of the ADC voltage when the sample and hold capacitor is connected. It takes around 500 ns for the voltage on the ADC pin to recover.

To maintain a consistent REF voltage reference value, take the sample after the capacitor has been charged. The charging rate of the capacitor depends upon the load.

The sampling time has been modified to 24.5 cycles (720 ns as the ADC clock prescaler is configured to four by default).

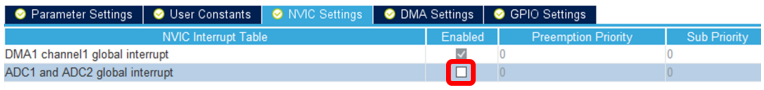
Note: See the Analog-to-digital conversion characteristics section of the product datasheet to adapt the sampling time to the actual impedance on the user board.

Figure 26. Scope plot of the ADC pin

In timing-critical applications, the performance of the ADC is optimized with a lower impedance at the potential divider.

The ADC1 interrupt is not used as the controller is automatically computed via the FMAC. The only use of the CPU is to update the PWM duty cycle with the value computed from FMAC.

Figure 27. **STM32CubeMX configuration of ADC1 interrupts**



After configuring the ADC via STM32CubeMX, the user must initiate the automatic calibration of the ADC in main.c before entering the infinite loop.

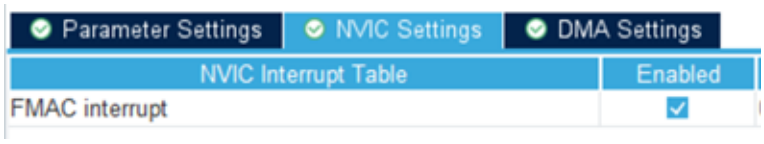
FMAC STM32CubeMX configuration

The FMAC is configured via the HAL library in the main.c.

However, the IRQ must be enabled, as it is used to update the PWM duty cycle each time a new value is calculated.

It is also possible to activate the IRQ directly from the FMAC section in the computing devices category of STM32CubeMX.

Figure 28. **STM32CubeMX FMAC IRQ configuration**



The FMAC is also initialized before entering the infinite loop, with the compensator coefficients declared in main.h.

After such configuration, the DMA destination for channel 1 requests can be associated to FMAC, and the ADC conversions can be started. This is performed in main.c before entering the infinite loop.

3.5.3 Program flow

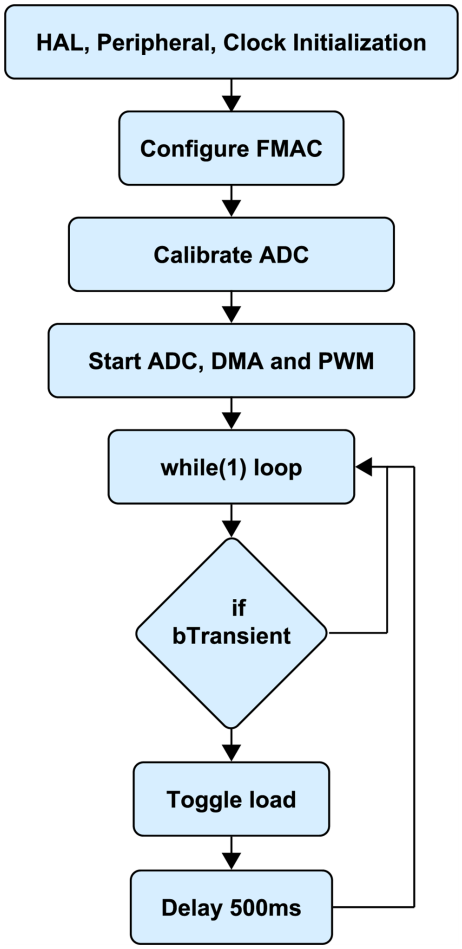
The main.c file contains most of the code for configuring and initializing devices. STM32CubeMX tool generates part of the code (see STM32CubeMX documentation on www.st.com).

Any additional code must be inserted between ***"user code starts here"*** and ***"user code ends here"*** comments inside the code files.

Outside of these comments, the code is automatically generated by the STM32CubeMX tool. If any changes are made to the project configuration within the tool, and the code is re-generated by clicking ***"generate code"***, the additional changes made outside these comments are lost.

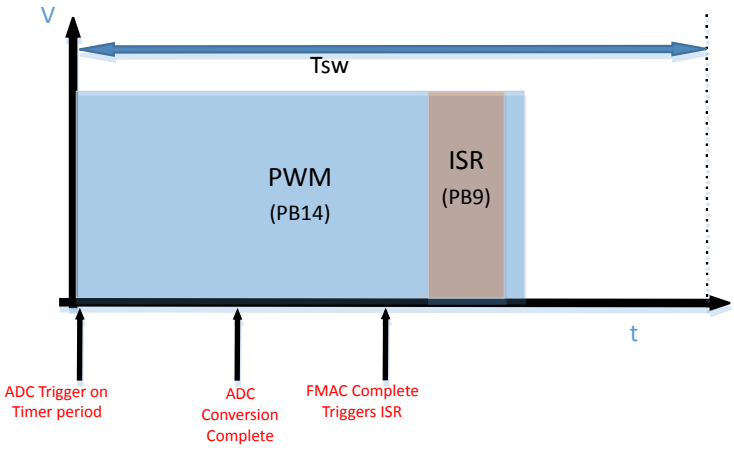
The structure of the main function within main.c is represented in Figure 29. This function contains calls to all initialization functions to configure the MCU peripherals. Next, the controller for the boost converter is set up and initialized with the calculated coefficients. Finally, the ADC is started and begins sampling the output voltage. The PWM outputs are enabled to drive the buck and boost switches.

Figure 29. Function flow of main() within main.c



The ADC module sampling is triggered by the HRTIM module (timer D period event). Once sampling and conversion are completed, the ADC activates DMA to copy the result directly from the ADC result log to the FMAC input register. This process is illustrated in Figure 30.

Figure 30. **ADC, DMA and FMAC timing diagram**



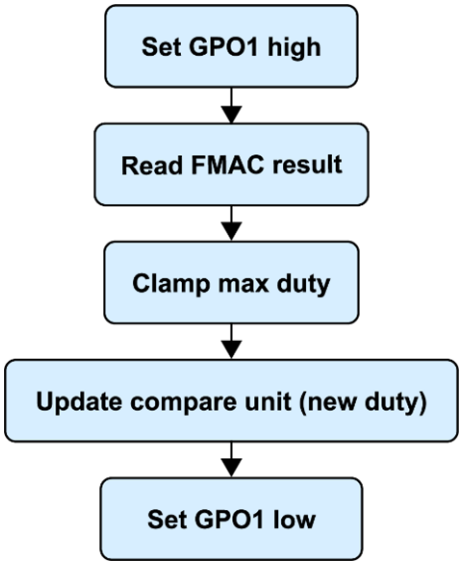
Interrupt service routine

After FMAC has completed running the 3p3z controller, it triggers an interrupt. This causes a jump of the MCU. The FMAC ISR is a separate function found towards the end of the `stm32g4xx_it.c` file included in this project. Its function name is `FMAC_IRQHandler (void)`.

The purpose of this FMAC ISR is to perform limit checking on the 3p3z controller output performed using the FMAC. The controller output is the new duty cycle value. This value is written in the register of the comparison unit 1 of the HRTIM.

The comparison unit has a maximum value, so the logic shown in Figure 31 is implemented within the user code section of the FMAC ISR.

Figure 31. **Flow of the FMAC ISR**



3p3z controller coefficients

The coefficients of the 3p3z controller are defined in `main.h`, at the top of the `main.c` file, by right-clicking on `main.h` and selecting `Open main.h` from the menu.

This header file contains several definitions for the pin names automatically generated by STM32CubeMX.

Further down this file, there is a section `/ * USER CODE BEGIN Private defines * /` where the definitions for the FMAC configuration start. The controller coefficients are defined under the FMAC configuration parameters. These coefficients (B0, B1, B2, B3, A1, A2, A3) are given in fixed-point hexadecimal form.

4 Application execution

The STM32 MCU comes with sample software that performs other functions such as controlling the RGB LED. To run the project, the user must compile the `Boost_VoltageMode_HW` project and flash the MCU. The minimum requirements are:

- A PC with Windows® 7 or later
- An STM32 compatible IDE, such as STM32CubeIDE or IAR Embedded Workbench
- STM32CubeMX (from v5.6.0), together with STM32Cube firmware library for STM32G4 Series (from v1.2.0), installed

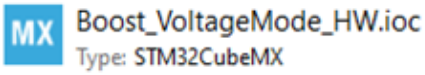
To get started:

1. Connect the micro-USB cable from the PC to CN3 on the kit
2. Apply power via the USB Type-C and connect this to CN2 on the kit
3. Ensure that the jumper JP1 is in the USB PD-VIN position

4.1 Loading the project with STM32CubeMX

Open STM32CubeMX by clicking on the *`Boost_VoltageMode_HW.ioc`* as shown in Figure 32.

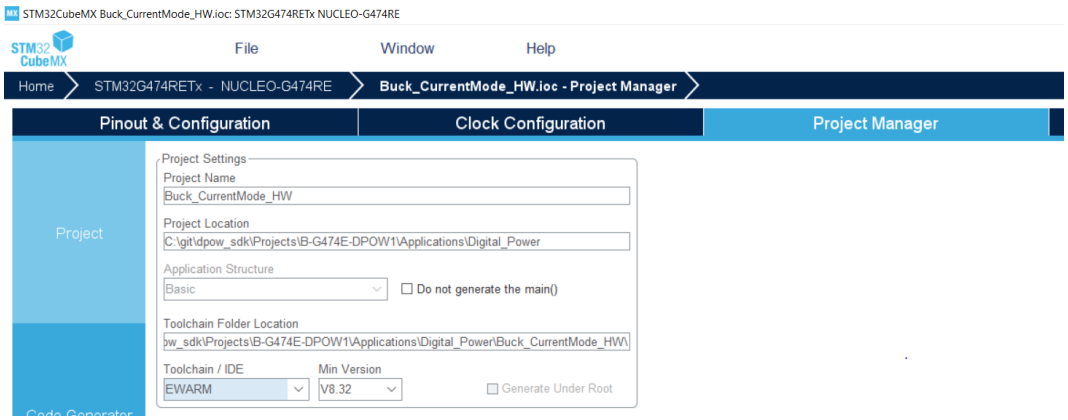
Figure 32. STM32CubeMx icon



4.2 Generate application code for IAR Embedded Workbench

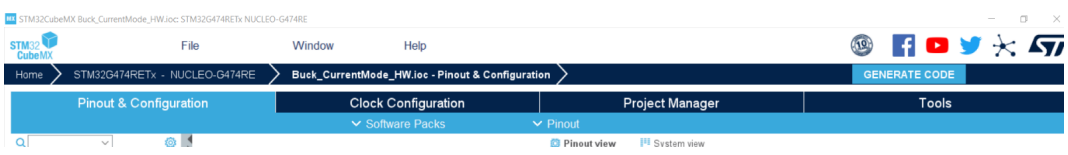
1. Once STM32CubeMX is opened, go to `Project Manager` panel, and select `EWARM` in `Toolchain/IDE` drop-down box.

Figure 33. Project manager setup



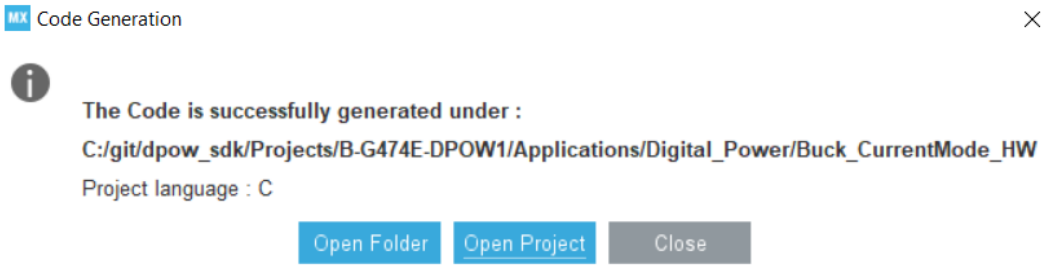
2. Then generate the code by clicking the `GENERATE CODE` button.

Figure 34. Code generation launch



3. Open the project by clicking `Open Project`.

Figure 35. Project opening after code generation



4. Right click on the IAR project and click `Options`, then STLINK, and check that `Emulator` drop-down box is on ST-LINK/V3.

Figure 36. IAR Embedded Workbench project options panel

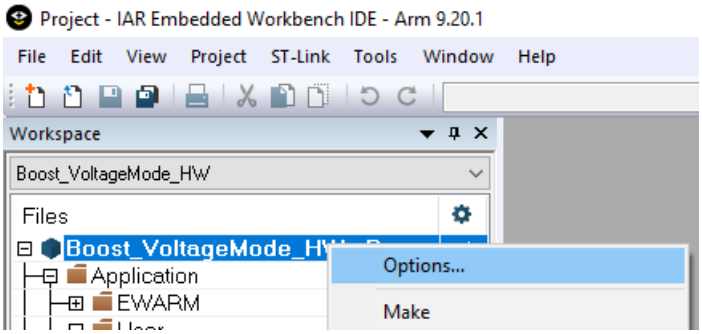
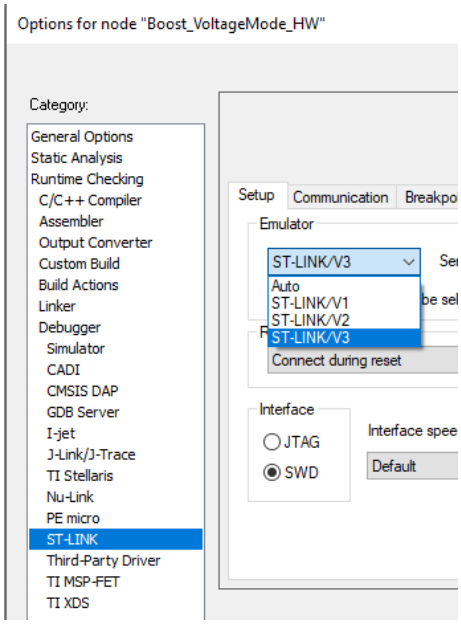
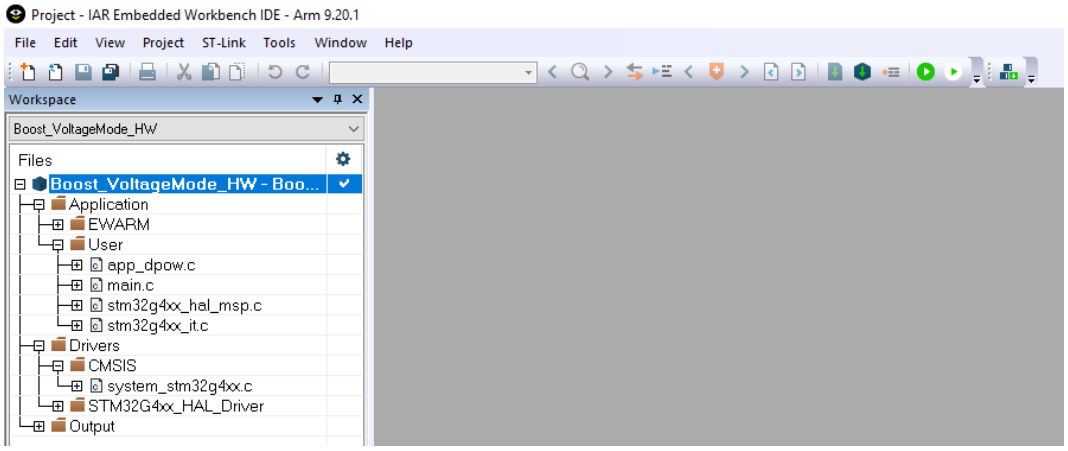


Figure 37. STLINK version selection



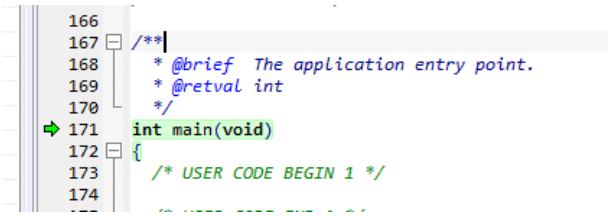
5. Inside the IAR™ IDE, click the `Download and Debug` icon that compiles and downloads the code to the MCU

Figure 38. Code compilation and downloading



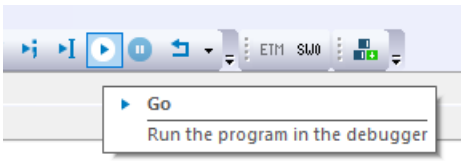
6. When the code has been downloaded on the MCU, it starts at the beginning of the “int main(void)” function.

Figure 39. Program counter set for running



7. To run the code, click the “Go” button.

Figure 40. GO button on IAR Embedded Workbench



Other buttons can be used to debug code:

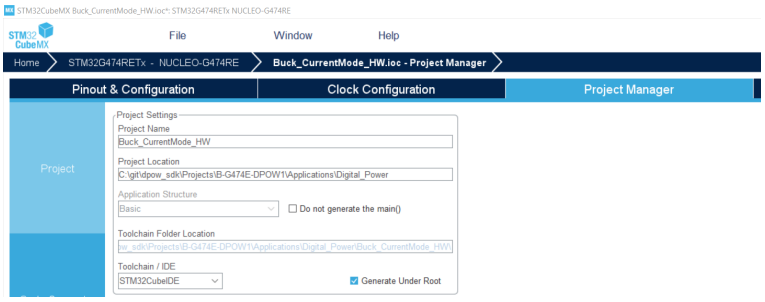
- **Break**: halts the code
- **Stop debugging**: terminates the debug session
- **Reset**: resets the code to the beginning and restarts

1. Click on stop debugging to end the debug session.
2. The firmware is downloaded into the microcontroller flash memory. IAR can now be closed if the debugging features are not being closed. The same program restarts each time power is applied to the kit as it is running from the MCU flash memory.
3. Close IAR to stop the debugger. If IAR asks to *terminate the debug session*, click *OK*

4.3 Generate application code for STM32CubeIDE workbench

1. Once STM32CubeMX is opened, go to *Project Manager* panel and select *EWARM* in *Toolchain/IDE* drop-down box. Select the repository where to generate the code by indicating it in the *project location* box.

Figure 41. Project manager setup

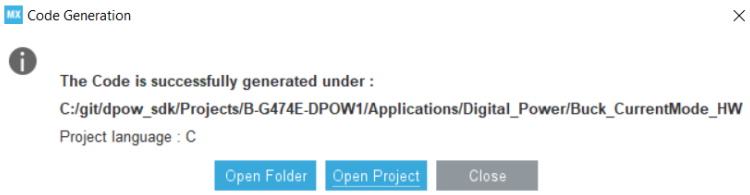


2. Generate the code by clicking the *GENERATE CODE* button.

Figure 42. Code generation launch

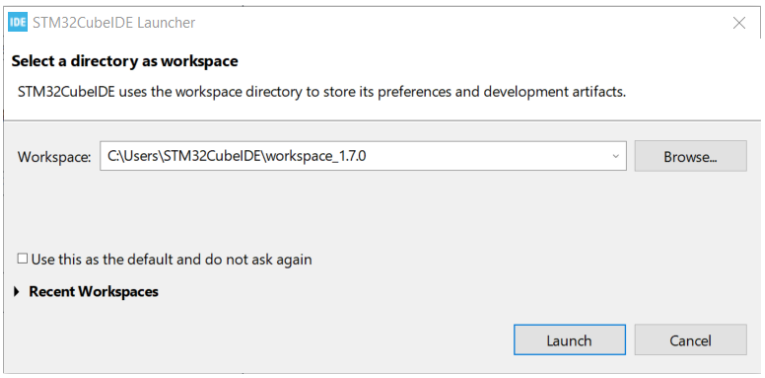
3. When prompted with the dialog box, open the project by clicking *Open Project*.

Figure 43. Project opening



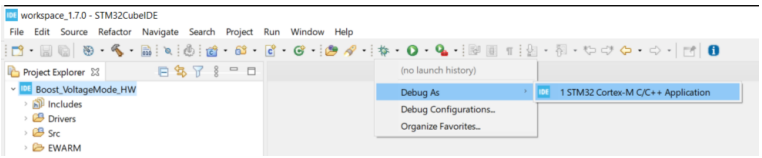
4. When the STM32CubeIDE launcher appears, select your workspace and click *Launch*.

Figure 44. STM32CubeIDE launcher



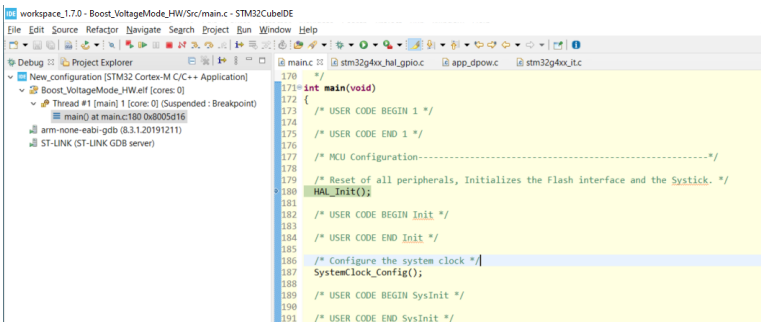
5. When STM32CubeIDE is opened, click the debug button, then *Debug As* and select *STM32 Cortex-M C/C++ Application*.

Figure 45. STM32CubeIDE code compilation and debugging



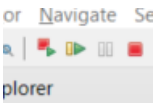
6. Code is compiled and downloaded into the MCU. The code starts at the beginning of the `int main(void)` function.

Figure 46. Program counter set for running



7. To run the code, click the *Resume* button.

Figure 47. Run button on STM32CubeIDE



Other buttons used to debug code:

- **Suspend**: halts the code
- **Terminate**: terminates the debug session
- **Terminate and relaunch**: resets the code to the beginning and restarts

1. Click on *Terminate* to end the debug session.
2. The firmware is downloaded into the microcontroller flash memory. STM32CubeIDE can now be closed if the debugging features are not being closed. The same program restarts each time power is applied to the kit, as it runs from the MCU flash memory.

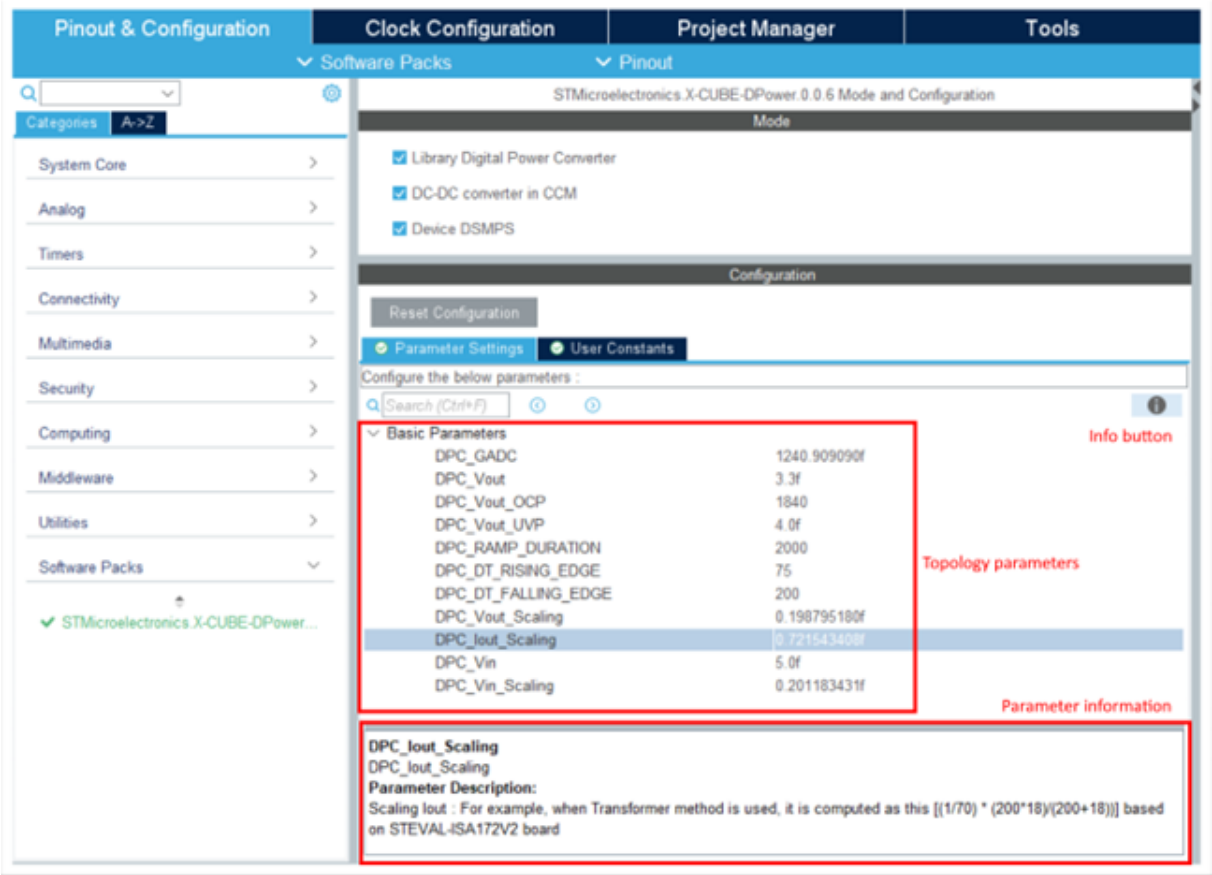
4.4 Boost converter using X-CUBE-DPOWER

Refer to X-CUBE-DPOWER user manual to install the software and to start-up with the desired converter.

4.4.1 Parameters configuration

When using X-CUBE-DPOWER, the configuration parameters are accessible through the graphical user interface. Their description is available either using the Info button (and then selecting the parameter), or by clicking directly on them.

Figure 48. **X-CUBE-DPOWER graphical user interface**



4.4.2 Driving the converter

To interact with the user, the B-G474E-DPOW1 board features a joystick and four color LED indicators.

Note: The buck converter generated from the X-CUBE-DPOWER, redefines the user's interaction. Note that is behaves differently from the standard G4 IntroPack example.

Use the joystick to command the converter:

- Up button: activates automatic load transients toggling
- Down button: deactivates automatic load transients toggling
- Right button: increase the total of activated load resistors
- Left button: decreases the total of activated load resistors
- Center button: unused

The LEDs inform about the converter status (refer to):

- Green: the system is running
- Red: an error or a fault has been detected
- Orange: not meaningful during automatic mode activation, reflects only the total of activated load resistors during manual mode activation:
  - Off: no load resistors
  - 1 flash: 50% load resistors
  - 2 flashes: 100% load resistors
- Blue: reflects the mode activation:
  - On: automatic mode
  - Off: manual mode

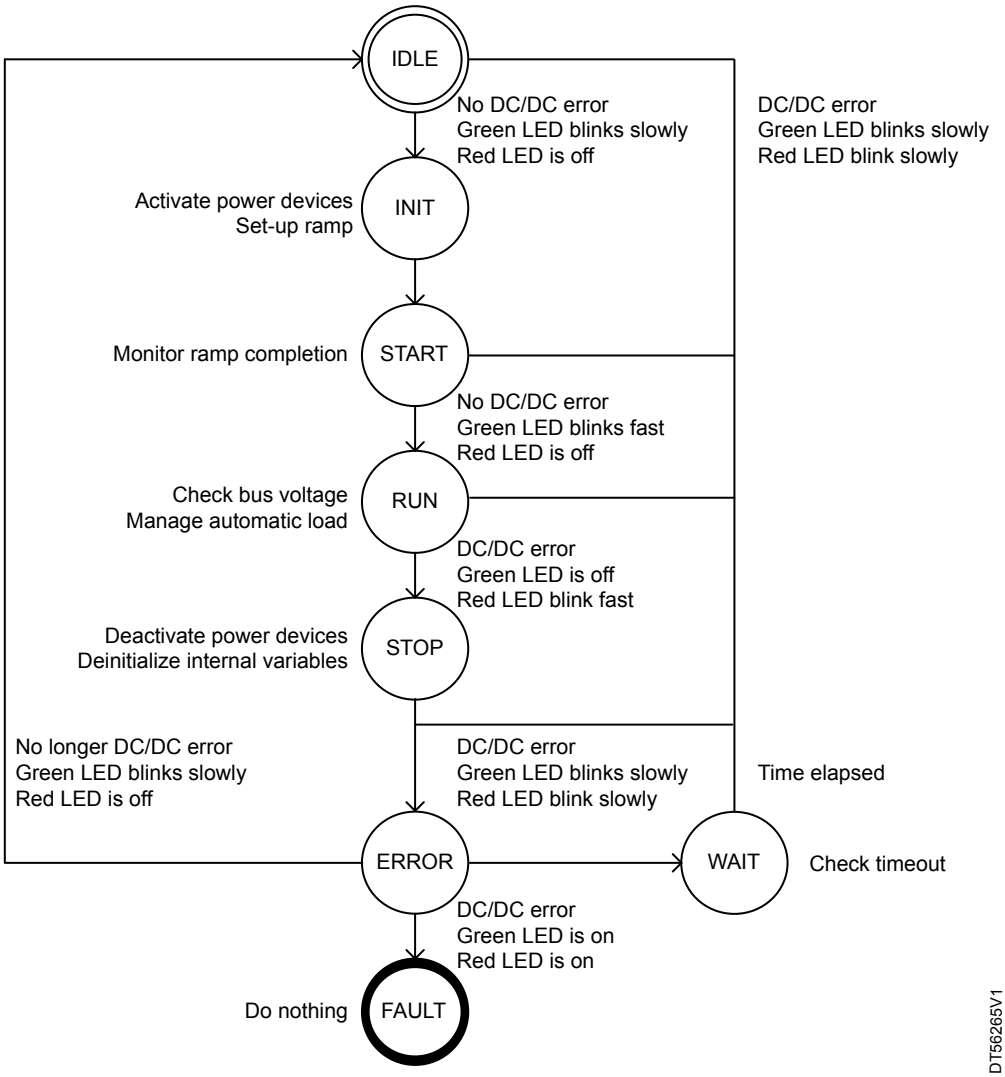
4.4.3 Dedicated files and state machine

Refer to X-CUBE-DPOWER user manual for details on the project files architecture and usage.

The “app_dpower.h” implements only the following user modalities (refer to the header file for a detailed description):

- #define OVERCURRENT_PROTECTION
- #define OVERVOLTAGE_PROTECTION
- #define SHORT_CIRCUIT_PROTECTION
- #define OVERTEMPERATURE_PROTECTION
- #define DEBUG_MODE
- #define DEBUG_COMP_OUT
- #define RUN_OPEN_LOOP
- #define PLOT_WAVEFORM

Figure 49. State machine



5 Results of measurements

5.1 Load regulation

The boost converter operates under closed-loop control. The converter responds to load changes, and regulates the output voltage to keep it constant. This is possible by monitoring the output voltage on the oscilloscope, and varying the load using the joystick. The duty cycle is also monitored, it changes slightly between the different load steps, due to losses within the power stage.

Figure 50 shows the output voltage and PWM for 0% load with the integrated load banks disabled. The oscilloscope measures the output voltage with an average value of 5.03 V, and a steady-state duty cycle of 52.2%.

The load increases to 50% of the rated power. The output voltage is 5.009 V, and the duty cycle increases to 63.2%. When the load increases to 100% of the rated output current, the output voltage value is approximately 4.96 V, and the steady state duty cycle increases to 72.2%.

Figure 50. Ouput voltage and PWM for 0% load

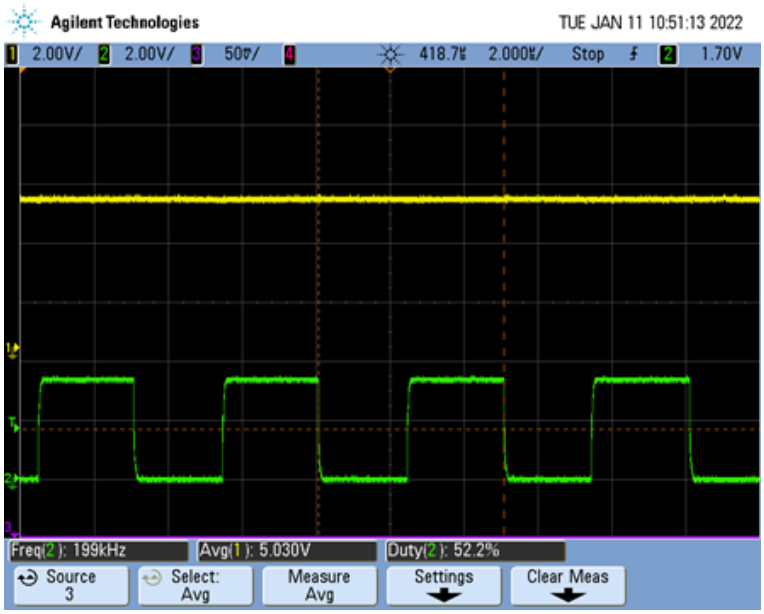


Figure 51. **Output voltage and PWM for 50% load**

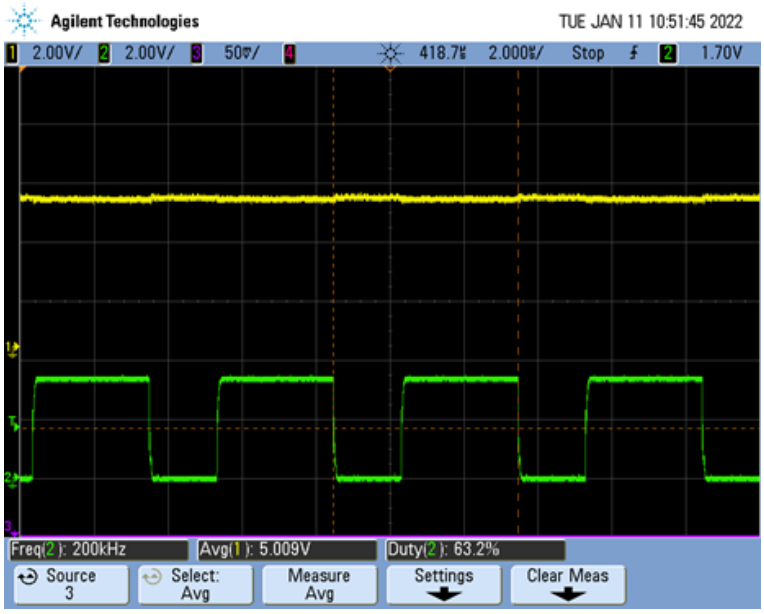
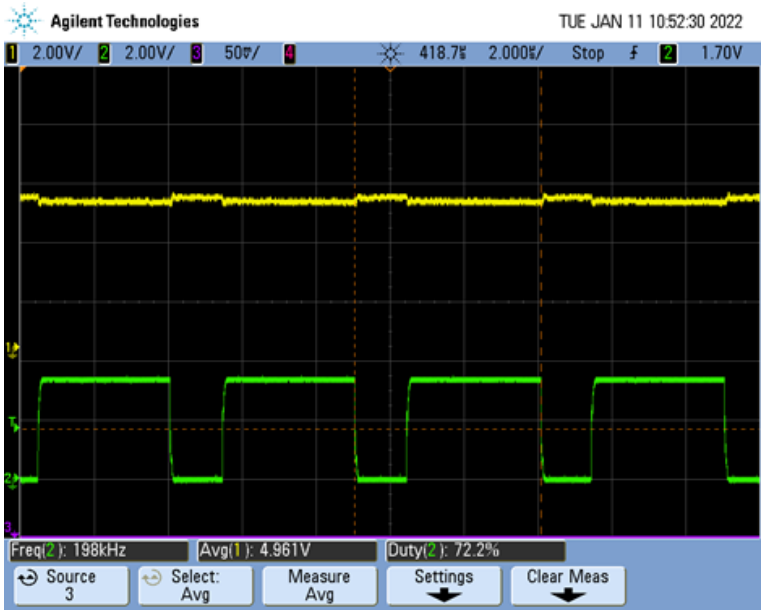


Figure 52. **Output voltage and PWM for 100% load**



As anticipated, the low performance of voltage mode control for boost topologies is seen in the figures above. The steady state output voltage changes slightly with the load steps. This becomes more evident in the transient response, where the effect of 400 kHz PWM drives the buck stage.

5.2 Transient response tests

The transient response provides useful information about the stability of the closed-loop system. It is measured by placing one oscilloscope channel on the output voltage, and another on the test point associated with the switching onboard load.

The output voltage channel is AC coupled, to see the deviation from the setpoint at the moment of the load transient.

Figure 53. Output voltage transient from load change 50% to 100%

Output voltage undershoot = 137 mV, settling time = 500 µs

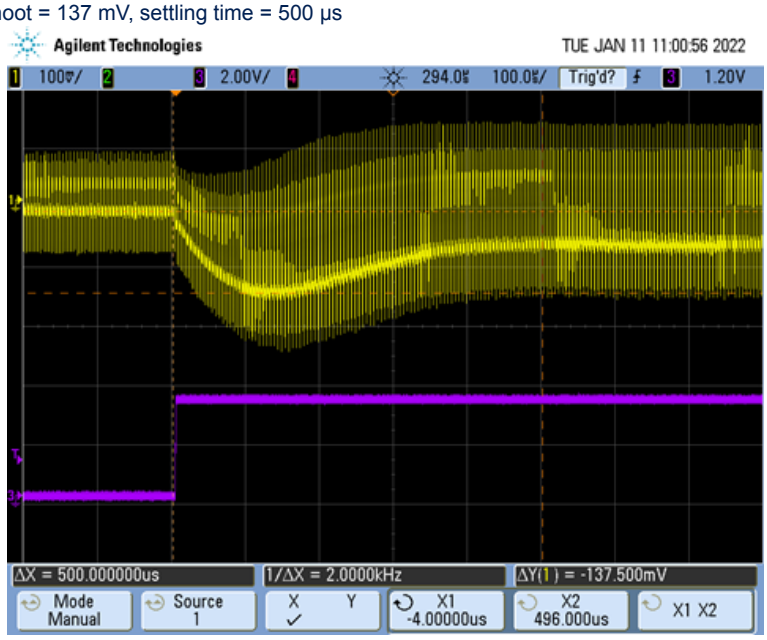
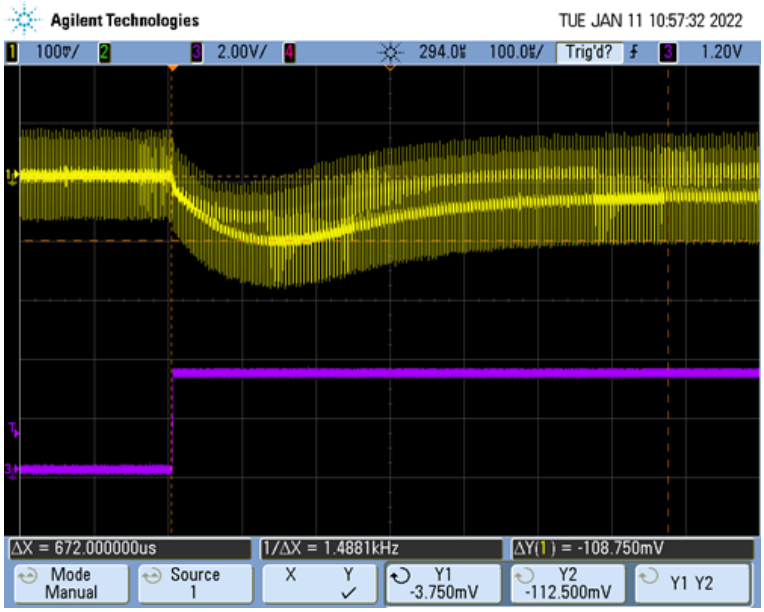


Figure 53 shows the transient response for a 50 to 100% load change. The output voltage on channel 2 deviates from the steady state by 137 mV, and recovers back to steady-state within approximately 500 µs. There is no ringing in the recovery, and a sufficient phase margin in the loop to ensure stability.

It is necessary to control the transient response over a range of line and load conditions. Figure 54 shows the transient response for a gradual change in load from 0 to 50%. With a light load, the response is noticeably slower: the recovery takes approximately 670 µs, but without no signs of oscillation. This indicates that the system is stable.

Figure 54. Output voltage transient from load change 0 to 50%

Output voltage undershoot = 108 mV, settling time = 672 µs



5.3 ISR plots (featuring FMAC and CPU load benefits)

The controller implementation uses the FMAC module available on STM32G474 devices. The FMAC is a hardware module that runs the controller in a few system cycles, without consuming CPU bandwidth. An ISR is called when FMAC finishes the controller computation.

To measure times, a GPIO pin is set high at the input of the FMAC ISR, and set low at the end. The time from ADC activation to ISR FMAC interruption is measured in Figure 55. The ADC is activated on the PWM period event, on the rising edge of the PWM signal (channel 2).

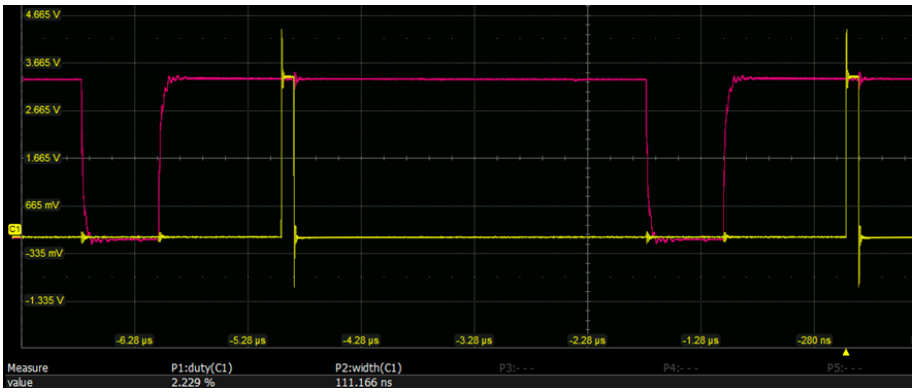
When the ADC is activated, the sample of the output voltage is converted and copied to the FMAC, using the DMA controller. When the FMAC is finished, the ISR is activated.

The sampling time is extended to 27.5 cycles, as explained in ADC1 STM32CubeMX configuration.

The FMAC ISR monitors the output limits at a minimum and maximum value. It also updates the counter comparison module register to set the new duty cycle value.

This ISR contains only a few lines of code, its duration is 110 ns (see Figure 55).

Figure 55. PWM and FMAC ISR duration



6 Conclusion

STM32G4 Series MCUs can be used to drive several PSU DC-DC converters with high-resolution PWMs, achieving very high duty cycle accuracy (down to 184 ps) with HRTIM1 IP. This IP contains several channels that drive different converters. Thanks to its highly configurable crossbar, several timer channels can operate with different configurations and inter-dependently, making it possible to target multiple topologies with a single MCU.

In this case, buck and boost switches are driven at different frequencies and duty cycles, although it is possible to build more sophisticated systems on the IP.

Thanks to the FMAC accelerator, the CPU load for the control circuit calculations is significantly reduced, freeing space for the rest of the application.

The voltage mode is not the control method intended to act on a boost converter. This application shows how to compute the coefficients of a compensator by measuring the frequency behavior of the plant.

Optimal control methods are the subject of workshops provided by Biricha, an STMicroelectronics partner.

7 Bibliography

1. AN5496, *Buck voltage mode with the B-G474E-DPOW1 Discovery kit*
2. C. P. Basso, "The Boost Converter," in *Switch-Mode Power Supplies*, McGraw Hill Education, 2014, pp. 45-54
3. Biricha, "Measuring the plant transfer function of a digitally controlled converter" workshop
4. Biricha, "Step-by-step Digital Power Supply Design with STM32" workshop
5. AN5497, *Buck current mode with the B-G474E-DPOW1 Discovery kit*

Revision history

Table 2. **Document revision history**

Date	Version	Changes
13-May-2022	1	Initial release.
09-Dec-2022	2	Added Section 4.4 Boost converter using X-CUBE-DPOWER and its subsections. Minor text edits across the whole document.
09-Jan-2023	3	Updated Section Introduction.

Contents

List of tables

List of figures