Control of a DC/DC Converter Using FlexPWM's Force-Out Logic

Implemented with MPC564xL

By: Yves Briant

1. Introduction

The MPC560xP and MPC564xL are two derivatives of the Qorivva family of automotive microcontrollers that have been tailored to easily control electrical machines like permanent magnet synchronous motors (PMSMs), brushless DC electric motors (BLDCs), DC/DC converters, and so on.

The FlexPWM module is one of the motor control peripherals embedded on these devices. Along with usual features like PWM generation, input capture, and so on, this module embeds some powerful mechanisms like force-out logic.

This application note describes how this force-out logic can be used to safely control a DC/DC converter with limited CPU usage. This control relies on other peripherals such as the eDMA, the ADC, and the eTimer. This document describes the configuration of each of these modules.

Contents

2. Control of a DC/DC Convertor

In the scope of a Hybrid Electrical Vehicle (HEV), a DC/DC converter is used for supplying the low-voltage network (12 V nominal) from the high-voltage traction battery (300 V, for example). The control method described in this document assumes the topology in the following figure.
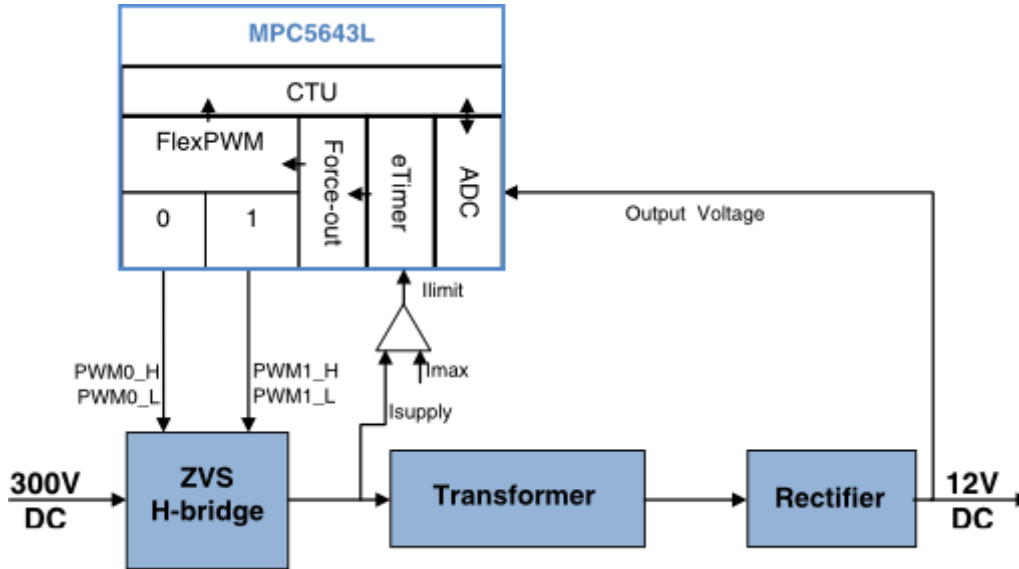

Figure 1. Typical topology of a DC/DC converter for HEV

The H-bridge works as a two-phase inverter. Controlled by the FlexPWM module, it has the capability to vary the amount of energy brought to the transformer. The transformer reduces the input voltage to a few tens of volts. The current supplied to the transformer is compared to a threshold. In case of an over-current, an Ilimit pulse is sent to the eTimer module linked to the force-out logic. Finally, the rectifier outputs a continuous DC voltage monitored by an ATD channel. The conversion result is used by software to pilot the FlexPWM module.

2.1. Basic control

The Zero-Voltage-Switch (ZVS) H-bridge is controlled with two pairs of phase-shifted PWM signals to keep the output voltage at the same level, in spite of variations in the input voltage and output current. *Figure 2* shows the simplified schematic of the converter and where the control is done.

The PWM0_H and PWM0_L signals on one side and PWM1_L and PWM1_H on the other side form two pairs of complementary phase-shifted PWMs. The period and duty-cycle of these signals are fixed (for example, frequency = 100 kHz, duty-cycle = 50 %). The amount of energy delivered to the transformer is achieved by varying the phase shift between the two pairs of PWM signals (see *Figure 3*).
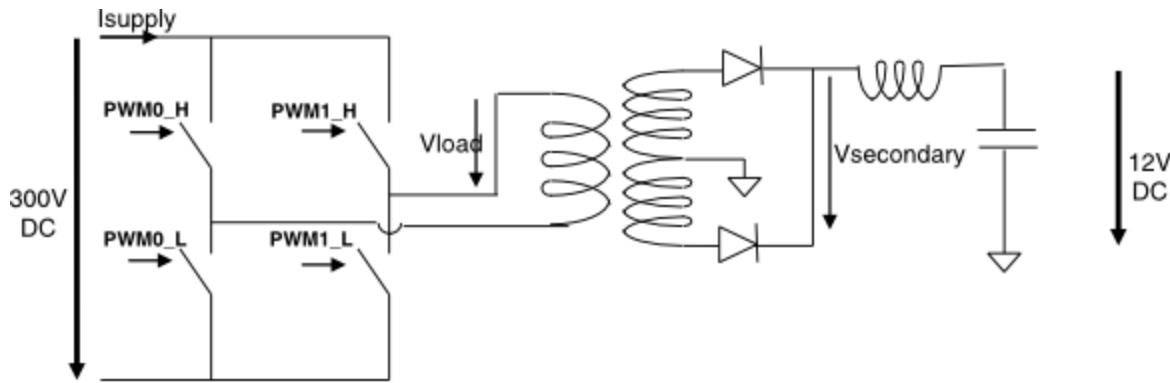
Figure 2. Simplified schematic of a DC/DC converter

Thus the control of such a DC/DC converter requires:

1. The generation of two complementary pairs of PWM, with deadtime insertion
2. An accurate control of the phase shift between the two complementary pairs

The flexPWM module offers these two features. The ''Phase-shifted PWM'' of MPC5643LRM, Qorivva MPC5643L Microcontroller Reference Manual is dedicated to the generation of phase-shifted PWM.
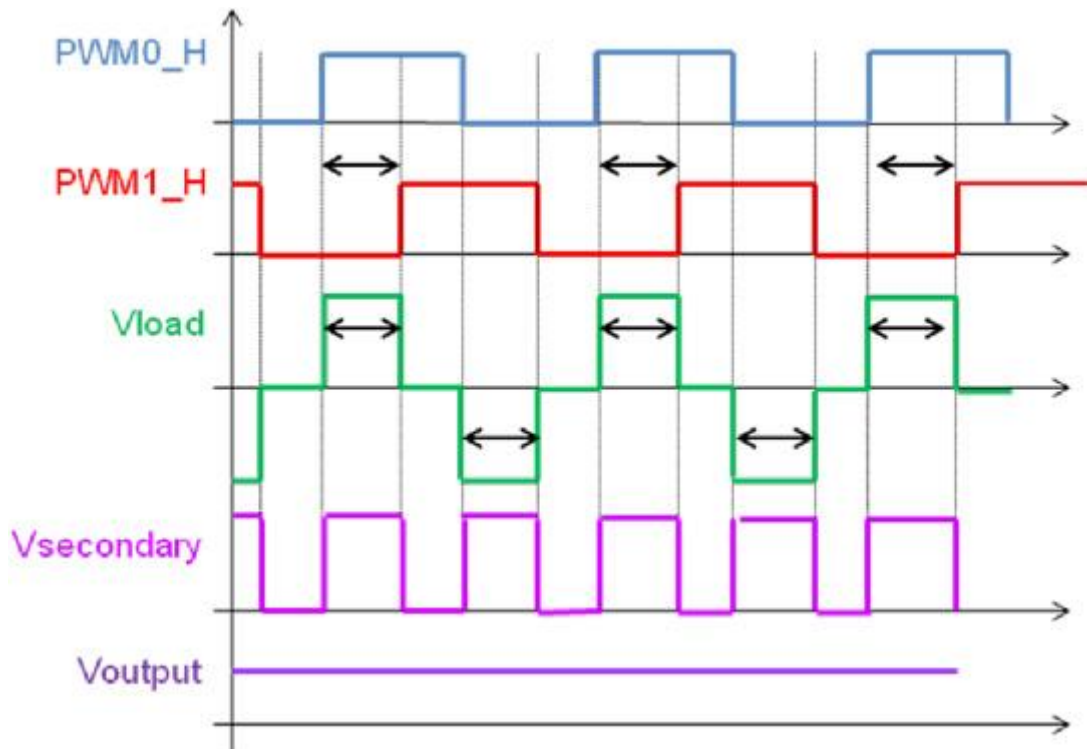

Figure 3. Phase shifted PWM and voltage at the primary of the transformer

2.2. Overcurrent control

The control of the DC/DC converter has to ensure that the current flowing from the high-voltage battery does not exceed a limit that could destroy the converter. An electrical problem within the transformer, the rectifier, or the load could be the cause for this overcurrent.

The reaction to this overcurrent should be as fast as possible (less than 1 µs) in order to prevent any electrical damage to the converter or to the high-voltage source. Thus, having a hardware reaction to this overcurrent (as opposed to a software reaction) would allow both a fast and a reliable control.

If an overcurrent is detected, the pair of PWM1 signals should be immediately set to the same level as the PWM0 signals, so that no energy is brought to the system. Thus, the phase shift between the two pairs of PWM signal is temporary reduced. This causes the Isupply current to decrease and the overcurrent condition to disappear.

Figure 4 shows the waveforms of the two pairs of complementary PWM signals, the Isupply current, and the overcurrent pulse. For sake of clarity the deadtimes are not shown on these waveforms. The dotted lines on the PWM1_H and PWM1_L show what would have been these signals without any reaction to the overcurrent.
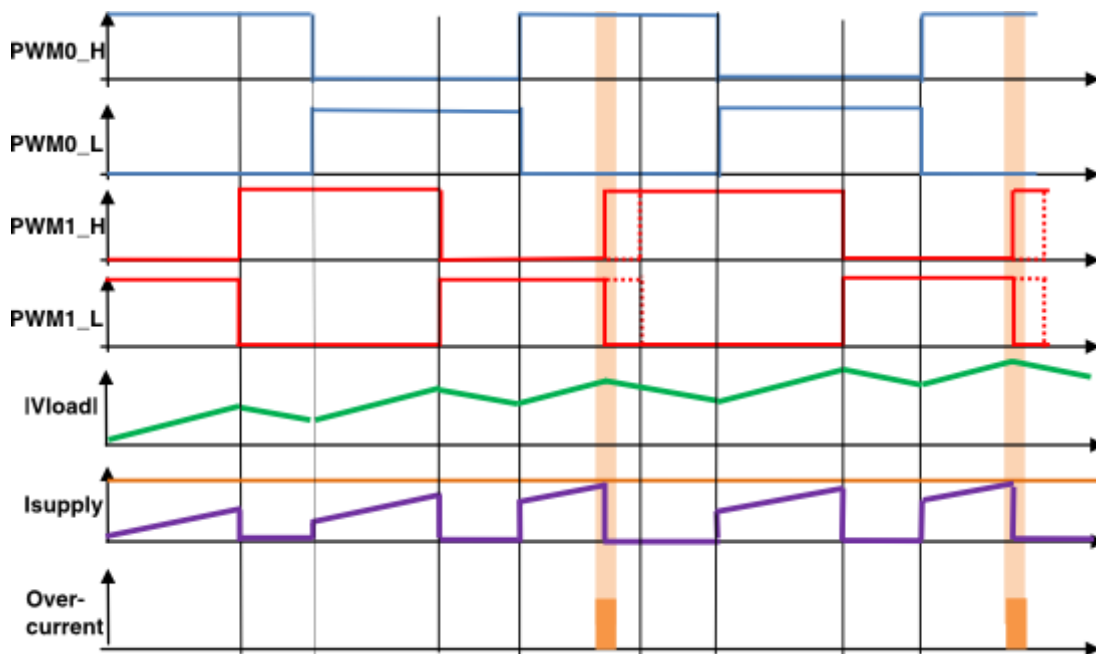

Figure 4. Overcurrent protection complements the state of the PWM0 pair

Section 3.3 Overcurrent control with force-out logic explains how this overcurrent protection can be implemented using the FlexPWM module's force-out logic, without any CPU load.

3. Implementation on MPC564xL

This section describes the implementation of this control and details the configuration of each peripheral used for this function.

3.1. Generation of the phase-shifted PWM

This use-case is already described in the MPC564xL reference manual. The first pair of PWM is generated by submodule 0 of one FlexPWM module, the second pair by submodule 1.The counters of these two submodules are configured with the same clock, initial, and overflow values and are started at the same time[1] so that they are always synchronized.

As already explained both PWM pairs have a fixed frequency of 100 kHz and a fixed duty-cycle of 50 %. Since only the second pair of PWM is modified to adjust the phase shift, the initial configuration of submodule 0 remains unchanged.

The frequency of the FlexPWM module is configured to be 120 MHz. The counters of both submodules are configured to count from the initial value -600 (0xFDA8) to the overflow value 600 (0x258), thus achieving a frequency of 100 kHz and a resolution better than 0.1 % for the phase shift.

Submodule 0 and submodule 1 are configured in complementary mode with automatic deadtime insertion: The DTCNT0 and DTCNT1 registers of submodules 0 and 1 have been set to the value 6, yielding a deadtime of 50 ns [2]. *Figure 5* shows how the 2 complementary pairs of PWM signals are generated. The grey area visible on each PWM are the deadtimes.

[1] Two counters of the same FlexPWM module can be started simultaneously by a unique write to the field RUN of the MCTRL register

[2] The duration of the deadtime mainly depends on the performance (switching time) of the transistors used to implement the bridge. 50 ns (or 0.5% of PWM period) is a typical value.
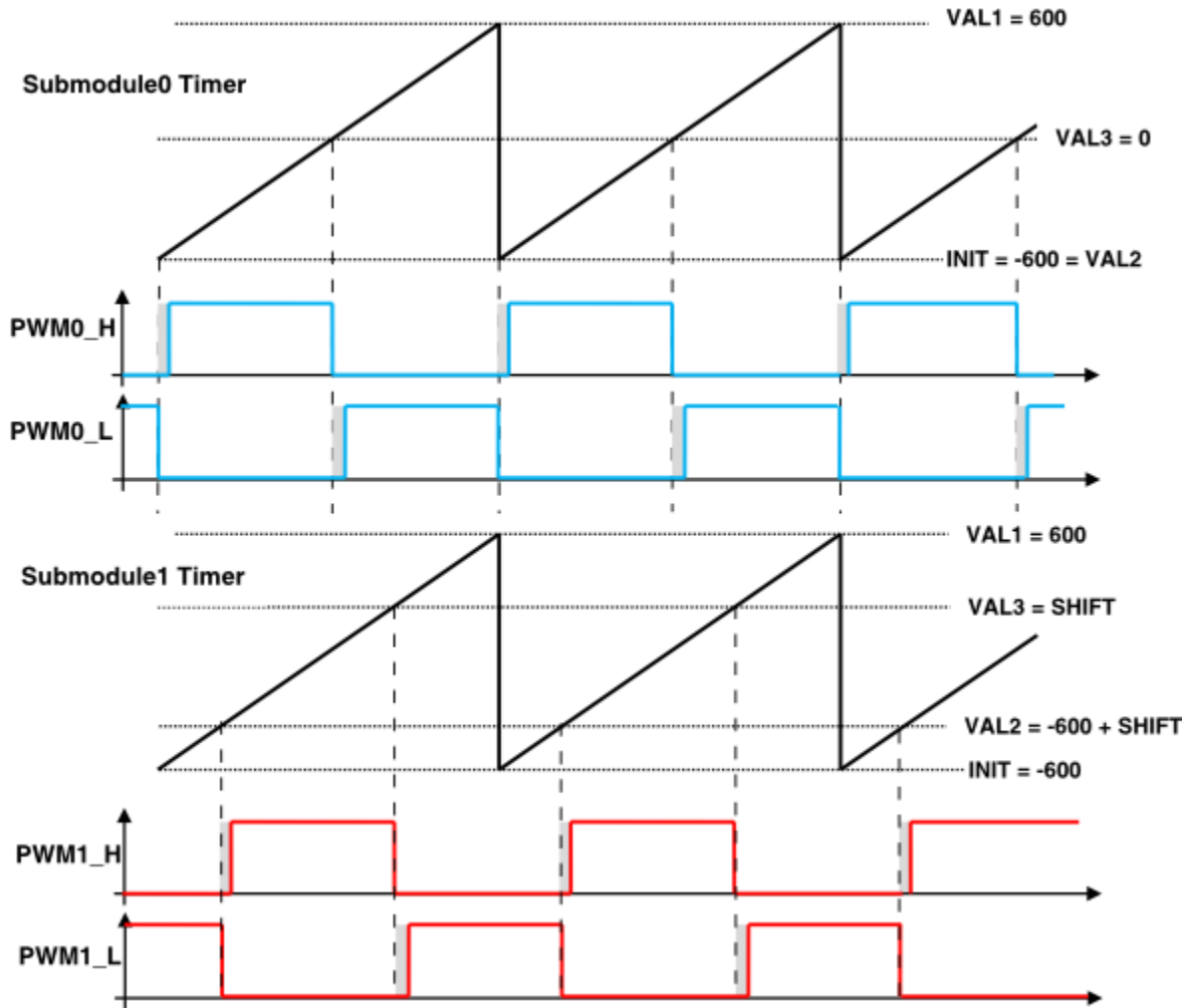
Figure 5. Generation of the phase-shifted PWMs with 2 submodules of the FlexPWM

The submodules 0 and 1 are configured for half reload. It means that any new values written to the VALx registers (among others) are taken into account when the timer reaches the value held by the VAL0 register. This is called a reload-event.

These reload events are also the triggers for the DMA transfers that are used to configure the force-out logic. This is further explained in *Section 3.3 Overcurrent control with force-out logic.*

3.2. Current acquisition with the CTU

The output voltage is regularly converted via an analog to digital channel[1], synchronously with the generation of the PWM0 signal. The CTU is configured for launching a single conversion exactly 7.5 µs after the rising edge on PWM0_H. The result is then stored in the FIFO0 embedded in the CTU. When the FIFO0 contains 4 ATD results, a DMA transfer is triggered that fills an internal RAM array. At the end of this DMA transfer, an interrupt is triggered. In this interrupt routine, the last four voltage readings are summed up and a new value for the shift is computed.

The diagram below illustrates how this chain happens:
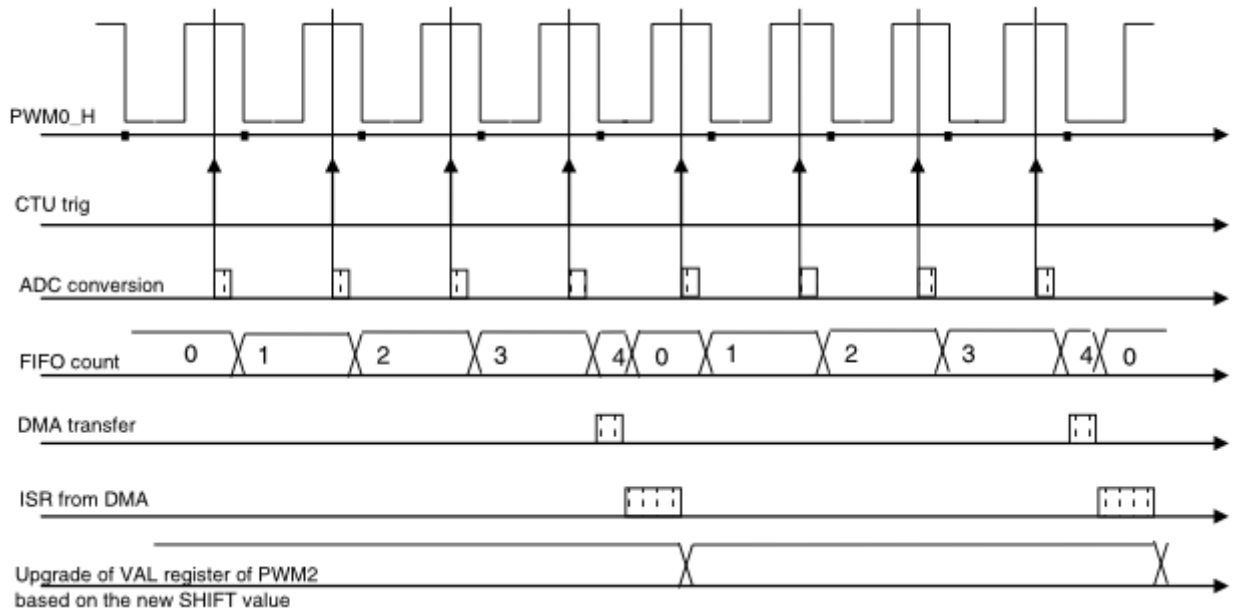
Figure 6. Current acquisition and update of the phase-shift

The shift value is computed in a simple way that could be greatly enhanced in a real application: the new shift value is inversely proportional to the measured voltage. A voltage of 0 generates a maximum shift of 600, and a voltage of 0x3FFF resets the shift value to 0.

Table 1. Calculation of the phase-shift

	Min	Max
One voltage reading	0	0xFFF (12bit conversion)
Sum of 4 voltage readings	0	0x3FFF
SHIFT value	600	0

[1] The 12 V output obviously cannot be converted directly. A resistor network divides the output voltage to a value within [0;5] V that the ATD converter can handle. Note also that the maximum output voltage should be taken in account when selecting the ratio of the resistor network.

It yields the following formula for computing the SHIFT value:

SHIFT = (0x3FFF – sum_4_voltages) * 600 / 0x3FFF

Which can be rounded to (in order to get rid of one division)[1]:

SHIFT = (0x3FFF – sum_4_voltages) * 37 /1024

The new SHIFT value is applied to PWM1_H and PWM1_L by adding it to the initial value of VAL2 and VAL3 in the submodule 1 of FlexPWM0.

CAUTION:

Modifying the double-buffered registers of a FlexPWM submodule is allowed only if MCTRL[LDOK] is reset. To clear this bit, MCTRL[CLDOK] should be used.

3.3. Overcurrent control with force-out logic

The configuration described in the previous section allows software control of the shift between the two PWM pairs, and thus of the output voltage of the DC/DC converter. The force-out logic of the FlexPWM should now be configured to automatically and immediately complement the second PWM pair in case of an overcurrent condition.

A full description of the force-out logic can be found in the FlexPWM chapter of the device's reference manual. One should consider this mechanism as a way to disrupt, permanently or temporarily, the continuous PWM generation.

Two things need to be configured when using the force-out logic:

1. The event causing this disruption, referred to as the force-event. It can be an internal event (reload event, overflow of the counter, or software command, for example) or an external event (EXT_FORCE signal). This setting is configured by the field CTRL2[FORCE_SEL] field of each submodule.

2. The signal to be applied on the PWM outputs when the force-event occurs. The possible configurations are:
   a) PWM signals are left unchanged —called *Normal mode* in this application note
   b) PWM signals are complemented—called *Complementary mode* in this application note
   c) PWM signals are set to a specific logic level, 0 or 1
   d) PWM signals are set to the level of an external signal

This configuration is achieved for each submodule by the DTSRCSEL and SWCOUT registers.

---

[1] Note that this rounding may not be acceptable in a real application, where the shift value needs to be computed and applied with a 0.1% accuracy.
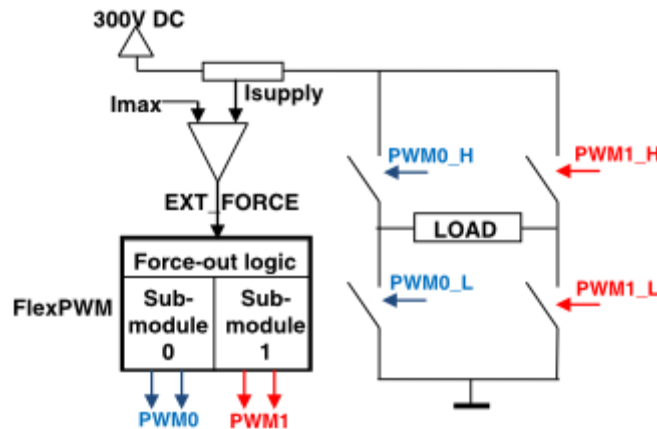
Figure 7. Connection of the FlexPWM module to the H-bridge and to the overcurrent signal

The principles of the overcurrent control with the force-out logic are the following:

- The force-out logic is switched into one of these three modes:

1. *Normal mode*: the PWM1 signals are left unchanged in case of a force-event

2. *One mode*: PWM1_H is set to 1, PWM1_L is set to 0 in case of a force-event

3. *Zero mode*: PWM1_H is set to 0, PWM1_L is set to 1 in case of a force-event

- At each falling edge of PWM0_H, the force-out logic is configured to apply Zero mode in the case of a force-event

- At each rising edge of PWM0_H, the force-out logic is configured to apply the One mode in the case of a force-event

- The force-event is configured to be the external EXT_FORCE signal, connected to the overcurrent signal.

- At each edge of the PWM0_H signal, the force-out logic is switched to Normal mode using a software FORCE. This action clears an optional overcurrent protection that would have occurred in the previous cycle.

- The configuration of the force-out logic at each PWM0 edge is done by the eDMA

Figure 8 illustrates these principles. Note that only the "high" PWM signal has been represented. The blue dotted lines on PWM0_H show the internal PWM0_H signal before the force-out logic.

Remarks:

- The configuration of the force-out logic performed by the eDMA at each edge of PWM0_H is only applied in case of an EXT_FORCE event, (that is, in the case of an overcurrent condition). As long as no over-current condition occurs, the force-out logic never leaves Normal mode.

- At each edge of PWM0_H, the force-out logic is switched back to Normal mode before being configured for the new setting (either One mode or Zero mode). This normal mode is applied by a so-called software force signal, (that is, by setting the MCTRL2[FORCE] bit).

- One may wonder why the "complementary mode" of the force-out logic has not been used. Indeed one may think this mode may remove the need for reconfiguration at each edge of PWM0_H. However consider the first overcurrent condition in *Figure 8*: if the force-out logic was configured in Complementary mode, the reaction to the overcurrent condition would have

Control of a DC/DC Converter Using FlexPWM's Force-Out Logic, Application Note, Rev. 2, 06/2016

been the same, but then the PWM1_H would have switched back to zero as soon as the timer reached the VAL3 values (edge represented by the dotted line). In any case the force-out logic needs to be reset to its Normal mode at each edge.

- As we have seen in the previous section, the DMA transfers are triggered by the "half-reload" events in the submodules 0 and 1, (that is, when the timers reaches the value hold by the VAL0 registers of each submodule). Thus the VAL0 values can be adjusted so that the DMA transfers are achieved exactly at the moment of the edges of PWM0. Typically VAL0 are set with the value 0 and 600 on submodule 0 and 1, so that the DMA triggers correspond to each edge of PWM0.
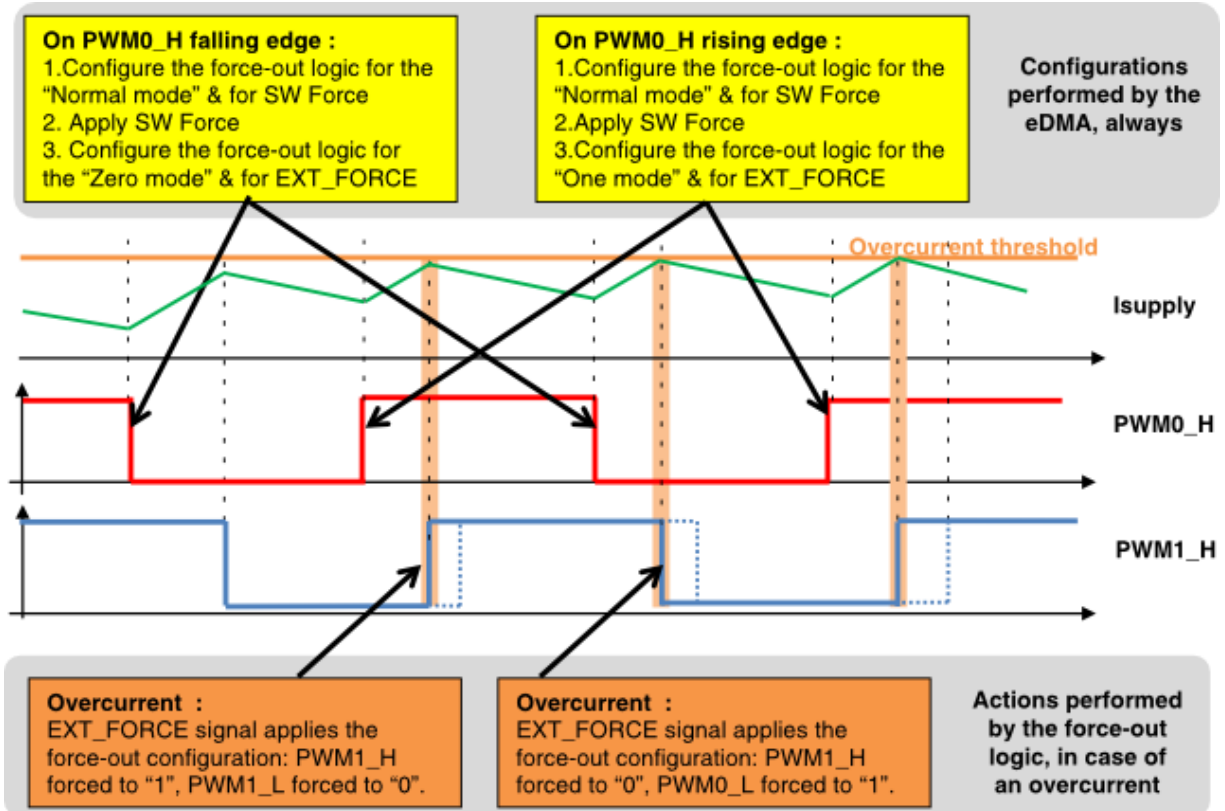

Figure 8. Principle of the overcurrent control with the force-out logic.

The register configuration should be the following at each falling edge:

1. Reset the force-out logic to its Normal mode:

DTSRCSEL[SEL23_1] = b00   (Generated PWM signal is used by the deadtime logic)

CTRL2[FORCE_SEL] = b000   (The local force signal is used to force the config)

CTRL2[FORCE] = b1   (Generate the local force signal by software)

2. Configure the force-out logic to its One mode:

DTSRCSEL[SEL23_1] = b10 (OUT23_1 bit is used by the deadtime logic)

SWCOUT[OUT23_1] = b1 (PWM0_H set to 1 in case of a force event)

CTRL2[FORCE_SEL] = b110 (The EXT_FORCE signal is used to force the config)

The register configuration is the same for the rising edges, with only one difference: SWCOUT[OUT23_1] is set to b0 instead of b1.

NOTE:

Since PWM0_H and PWM0_L are configured as a complementary pair of PWM signals, PWM0_L will be automatically complemented when the force-out logic inverts PWM0_H.

3.4. Force-out logic reconfiguration by eDMA

As described earlier, the register configuration described in the previous section should be performed by the eDMA at each edge (falling and rising) of the PWM0_H signal.

3.4.1. Generation of the eDMA requests by the FlexPWM module

When configured for PWM generation, the FlexPWM module can generate a DMA request at each reload event[1]. It means each time the timer reaches VAL0 if half reload is enabled, and each time the timer reaches VAL1 if full reload is enabled.

This trigger is called *VALx write request* in the reference manual. All the triggers from the four submodules are merged in a single trigger source to the DMA_MUX, which is called *comp_val*.

Within one PWM period, two *comp_val* triggers will be generated:

- One from submodule 0, when its internal counter reaches the value of VAL0 (half-reload), initialized to 0.
- One from submodule 1, when its internal counter reaches the value of VAL0 (half-reload), initialized to 600.[2]

3.4.2. Description of the eDMA transfers

According to *Section 3.3 Overcurrent control with force-out logic*, three registers (SWCOUT, DTSRCSEL, and CTRL2) need to be configured by DMA at each edge of the signal PWM0. Although the addresses of these registers are not contiguous, it is possible to implement the necessary configuration using only one DMA channel in one single transfer.

---

[1] A reload event is an event causing the double-buffered registers of the FlexPWM module to be loaded. A reload event is either the FlexPWM timer reaching the VAL0 value (half-reload), or reaching the VAL1 value (full-reload).
[2] Actually, the DMA transfers are triggered a little before the rising and falling edges to account for the delay of these transfers. By doing so, the force-out configuration is effective exactly at the moment of the edge.

The parameters of this DMA channel are as follows:

Table 2. eDMA channel parameters

| Parameter	Value
Number of bytes transferred (NBYTES)	64bits
Source Size (SSIZE)	64bits
Destination Size (DSIZE)	32bits
Destination Offset (DOFF)	0xFF10 (-0xF0)
Minor loop offset (MLOFF)	0x01E0
Destination address last adjustment (DLAST)	0x01E0
Number of iteration in a major loop (BITER)	2

The figure below shows how a transfer occurs within a minor loop:
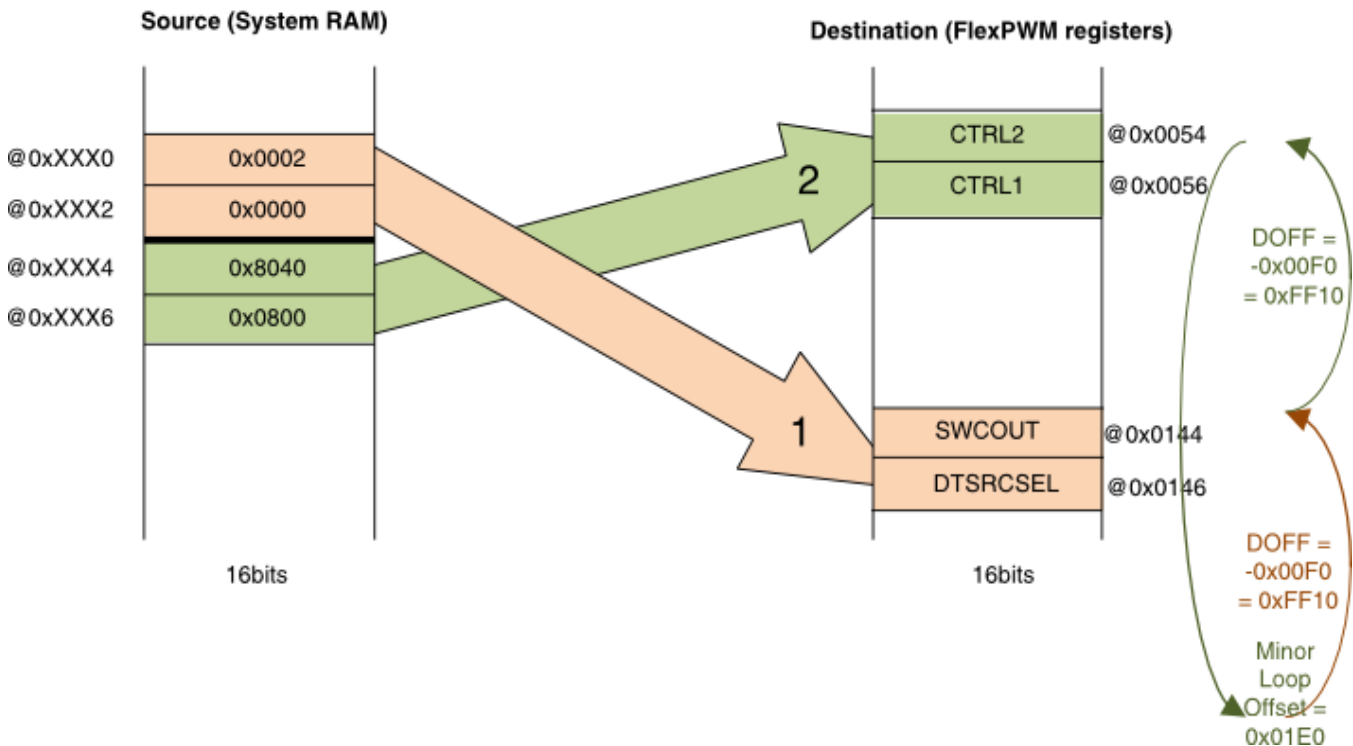
Figure 9. DMA transfer during a minor loop for Force-Out reconfiguration

DMA channel 0 is used to implement this transfer. As mentioned in *Section 3.3 Overcurrent control with force-out logic*, two configurations needs to be applied at each edge of the signal PWM0: the configuration for the Normal mode and the configuration for the One or Zero mode. This is achieved by channel linking: channel 0 links to itself at the end of each minor loop, and the iteration loop is set to 2.

In order to debug the DMA transfer and to assess their duration, an additional channel (channel 1) is configured to toggle a pin at the beginning and at the end of each transfer. *Figure 10* shows how channels 0 and 1 are linked to each other.
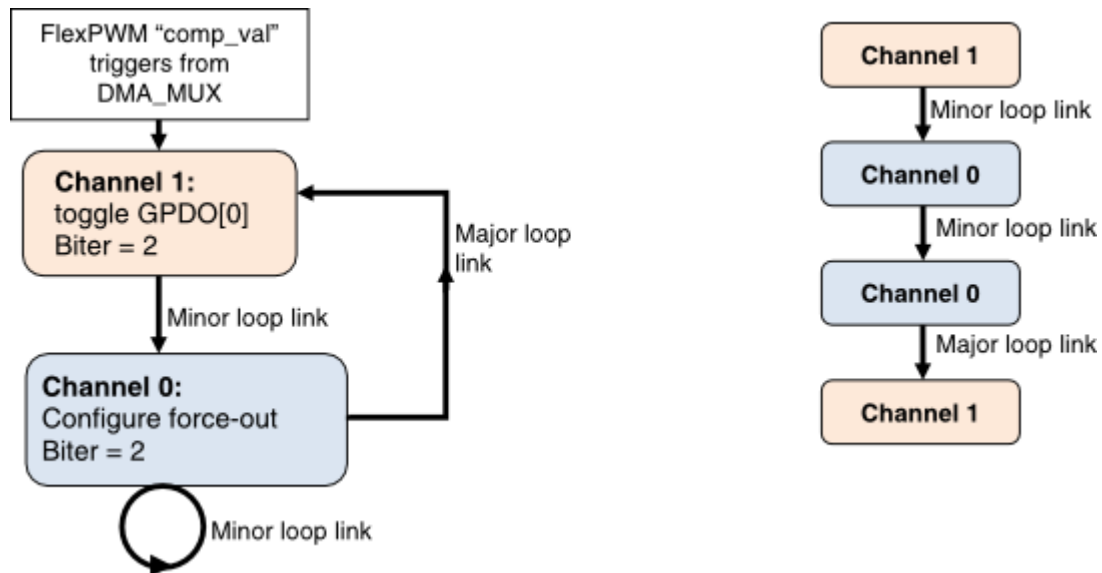
Figure 10. Configuration and linking of the eDMA channels for the force-out reconfiguration

Several points need to be noted regarding the transfer generated by DMA channel 0:

1. 64bits are transferred at each DMA trigger, but the destination size (DSIZE) is set to 32bits. This allows the 64bit transfer to be broken down into two 32bit transfers with different destination addresses. An offset of -0x00F0 (DOFF) is applied to the destination address after each 32bit transfer.

2. The CTRL1 register is written to as a side effect, but its configuration is not required.

3. At the end of the 64bit transfer (minor loop), the destination address need to be adjusted in order to point to CTRL2 again. This is achieved by enabling the Minor Loop Mapping capability on the destination address.

4. Each *comp_val* event from the FlexPWM starts two minor loops of channel 0. Each minor loop of channel 0 configures the force-out logic for a specific mode (Normal mode, Zero mode, One mode) in the following sequence:

5. Normal mode > One mode > Normal mode > Zero mode

6. The source memory for these transfers contains 4 groups of 64bits, each group corresponding to one mode: Normal, Zero or One. This is depicted in the following figure.
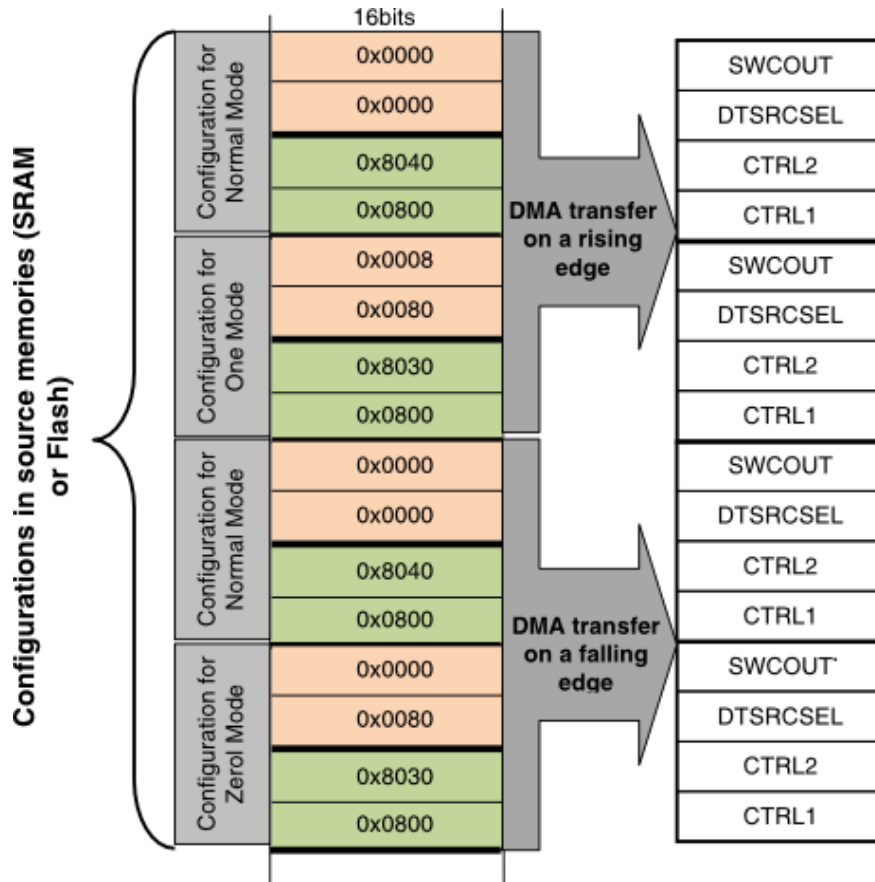
Figure 11.   Force-out logic configuration for Normal-mode, One-mode, and Zero-mode

3.5.   Connection of the limit signal to the EXT_FORCE input

The EXT_FORCE input of the FlexPWM module is not an external pin of MPC564xL. The EXT_FORCE signal is connected to the internal output of the eTimer0's the channel 1.
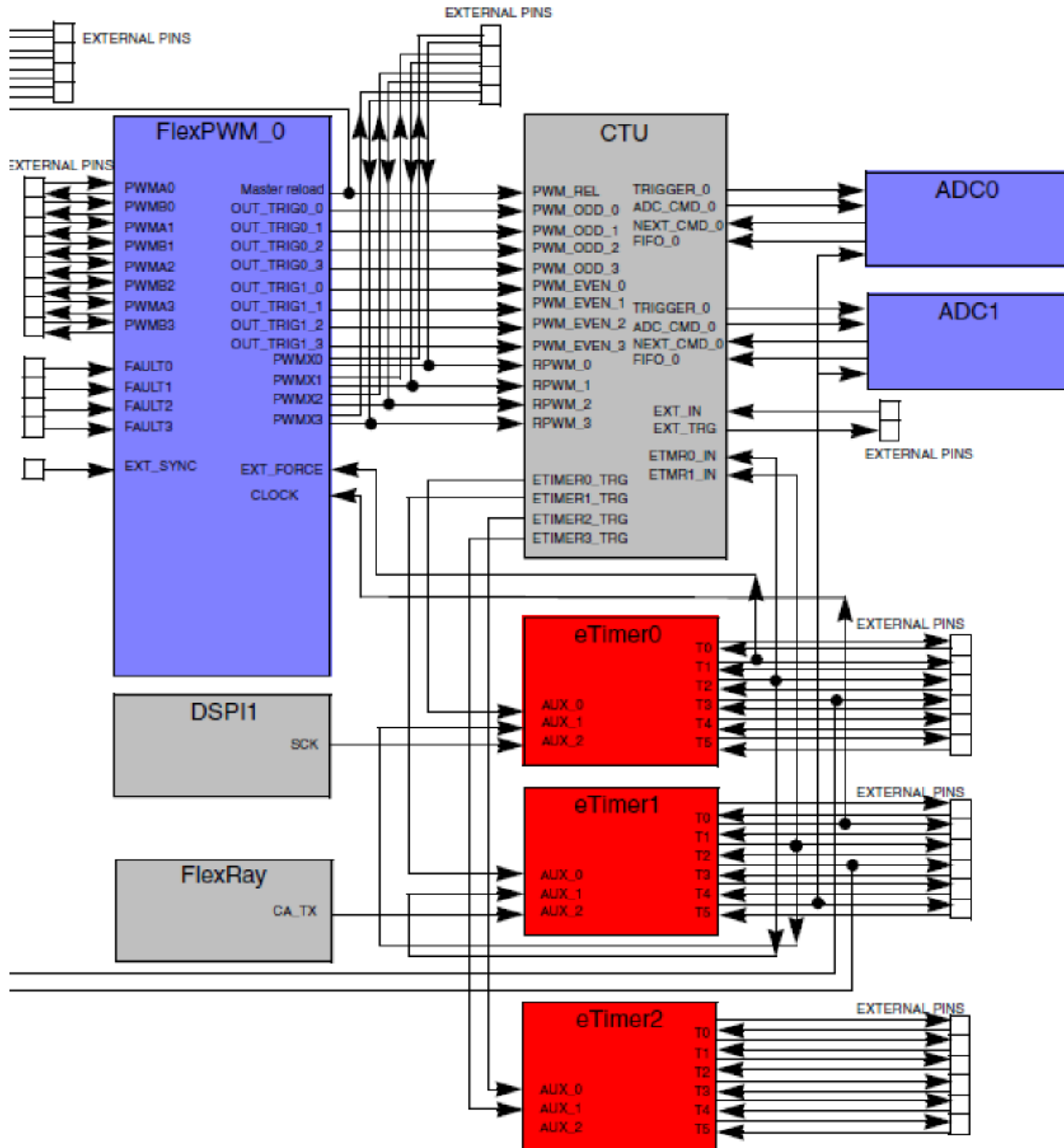
Figure 12. Connections of the motor control peripherals on Leopard

Channel 1 of eTimer0 is configured to work in EDGE-COUNT mode. In this mode, the channel 1 counter counts both edges of the primary source. The primary source is the channel 0 input pin, which is connected to the Ilimit. The output mode (selected with the CTRL2 register) is configured to enable gated clock output while counter is active. This mode is selected so that the channel 1 output flag (OFLAG1) mirrors exactly the overcurrent condition.
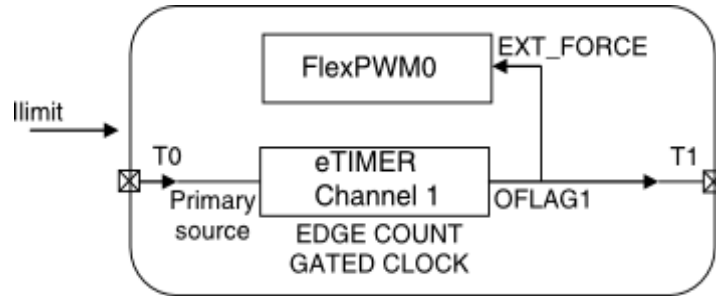
Figure 13.   External and internal connections of the Ilimit signal

4. Performance

The figure below shows the result of the implementation of this control.

The PWM0 signal's behavior conforms to what was expected: it is complemented as soon as the Ilimit pulse arrives.

The signal in pink shows the state of the pin used to debug the DMA transfer responsible for reconfiguring the force-out logic at each edge of PWM1.

NOTE

> The actual transfer time is shorter than the length of the pulse, which accounts for three transfer durations (two transfers to toggle the pin and one transfer for the force-out logic).
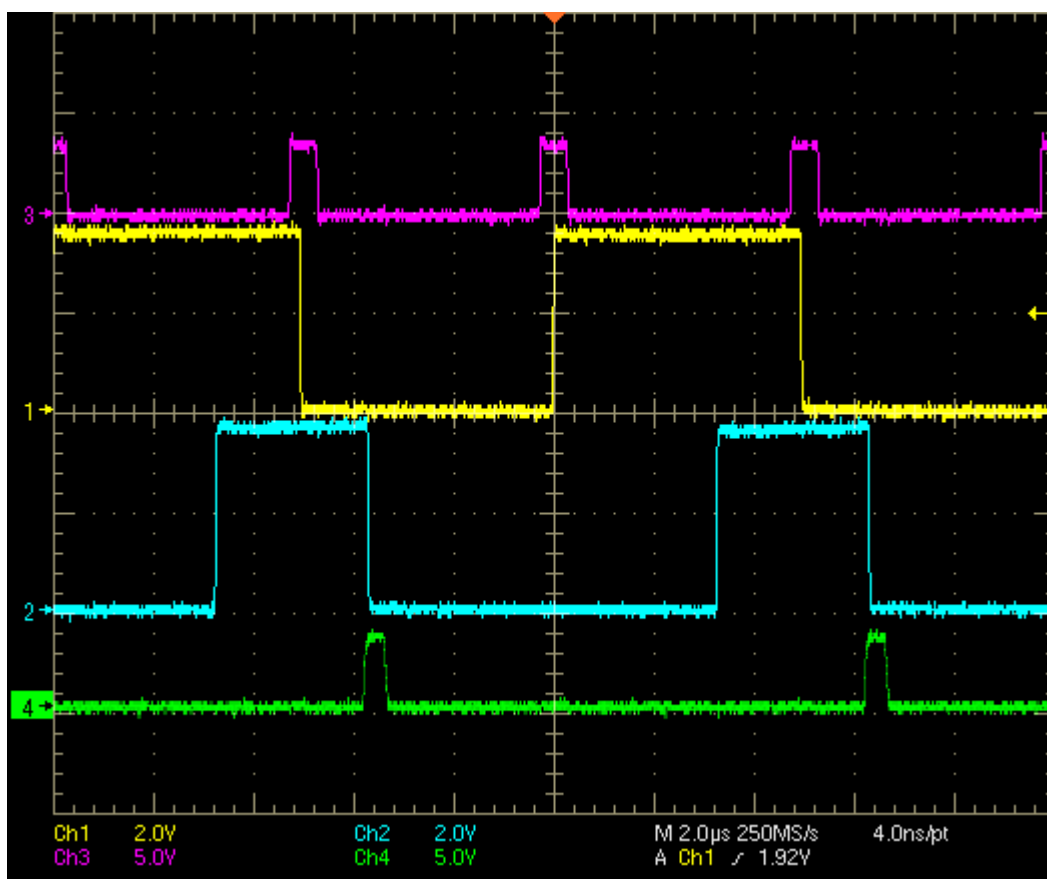
[1] Pink : DMA debug pin for force-out reconfiguration
[2] Yellow: PWM0_H signal (generated by FlexPWM0 submodule 0)
[3] Blue: PWM1_H signal (generated by FlexPWM0 submodule 1)
[4] Green: Ilimit signal connected to eTimer0 channel 1

Figure 14.   Observation of the PWM0_H and PWM1_H signals in case of an overcurrent

There are two critical timings to measure the performances:

1. The delay between the Ilimit edge and the action to complement the PWM1 signals: if this time is too long, an overcurrent condition may cause some damage before the bridge is put in a safe

state. This delay is called *reaction time*.

2. The minimum time between an edge of PWM0 and a subsequent Ilimit pulse. If an Ilimit pulse arrives too early after a PWM0 edge, the DMA transfer reconfiguring the force-out may not have the time to complete, and the PWM1 signal will not be complemented instantaneously. To minimize the delay, the DMA transfers can be started slightly before the edges of the PWM0 to account for the time required for the DMA to load the TCD and to achieve the second portion of the 64bit transfer that really loads the force-out configuration. To do this, the VAL0 value of the submodules 0 and 1 can simply be adjusted. This time is called *configuration time*.

4.1. Reaction time

The following diagram shows the delay observed between the Ilimit pulse and the complement of PWM_1.
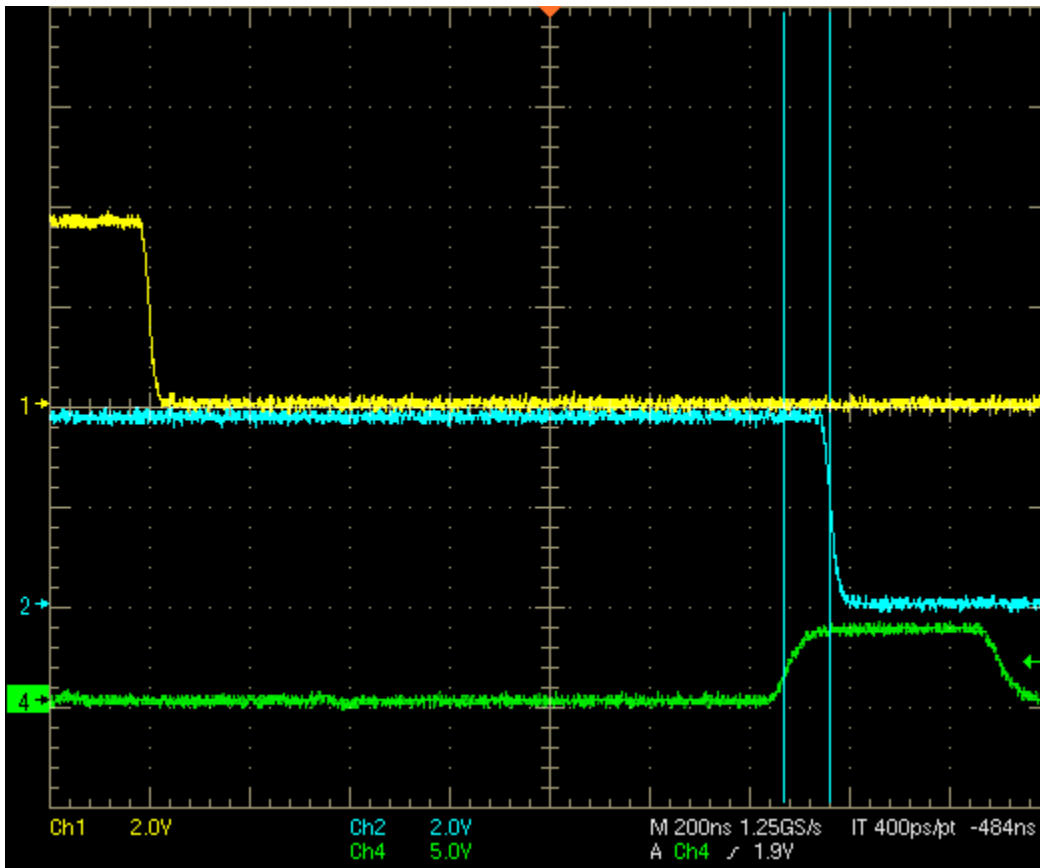
[1] Yellow: PWM0_H
[2] Blue: PWM1_H
[4] Green: Ilimit signal connected to eTimer0 channel 1

Figure 15.   Delay of the force-out logic activation

This delay has been measured to be less than 100 ns. This time is basically the reaction time of the force-out logic and the delay introduced by the eTimer channel gating the Ilimit signal.

4.2. Configuration time

At each edge of the PWM0 signal, the force-out is reconfigured via a DMA transfer. This transfer takes some time. During the transfer, if an overcurrent pulse arrives, the force-out will not react. Three points contribute to keep that configuration time as short as possible:

1. The DMA transfer for the force-out configuration should have the highest priority over other DMA transfers

2. The DMA should have the highest priority on the crossbar in regards to other bus masters like the cores or FlexRay. This highest priority has to be set for the two slave ports of the crossbar used by the DMA transfer: the SRAM and the PBRIDGES. The core can keep the highest priority on the flash memory. In addition, the parking option (the default master port connected to a slave port when the slave is not used) has an impact on this transfer time. The SRAM and PBRIDGE slave port should have the DMA as the default parking option.

3. The launch of the DMA transfers can be advanced by changing the VAL0 value of submodules 0 and 1. The following result has been obtained by launching the DMA transfer 300 ns before the edges of PWM1.
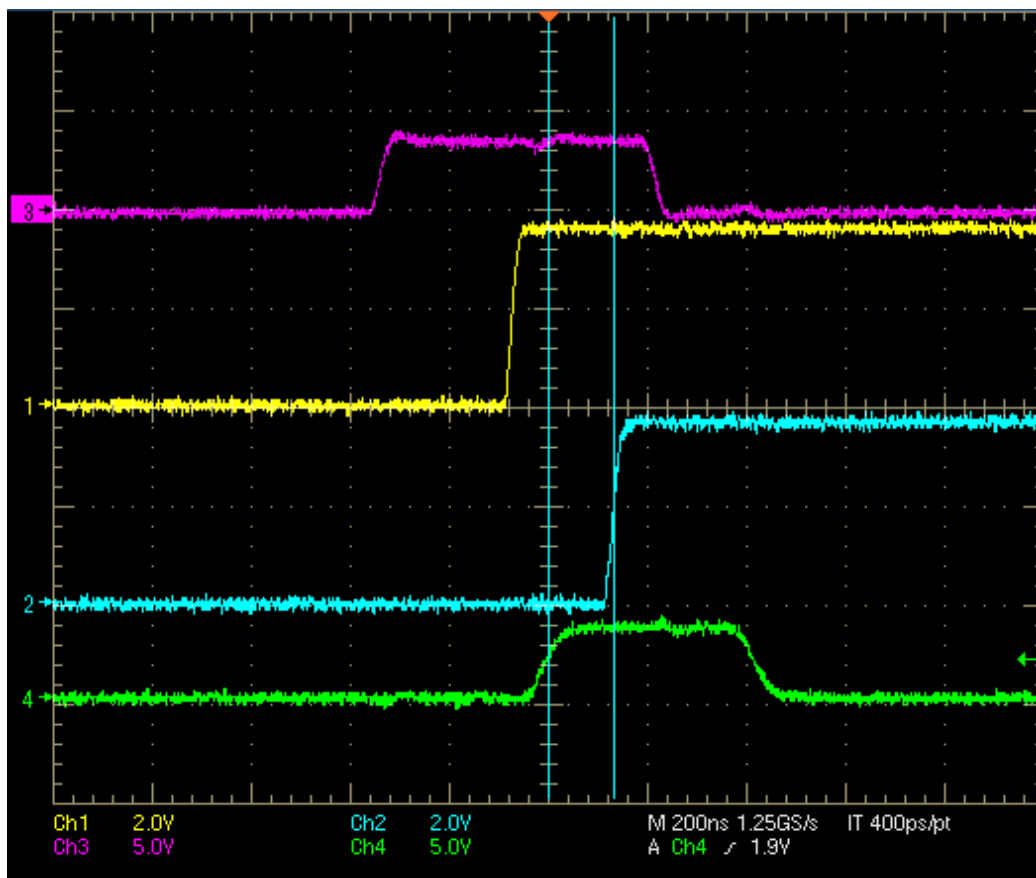
[1] Pink : DMA debug pin for force-out reconfiguration
[2] Yellow: PWM0_H signal [3] Blue: PWM1_H signal [4] Green: Ilimit pulse

Figure 16. Behavior of the PWM signals if the Ilimit signal appears before the force-out logic reconfiguration

It is important to note that the force-out logic is level-sensitive. So if the Ilimit pulse occurs before the DMA transfers complete, the new force-out configuration will apply as soon as the DMA transfer finishes.

This scope shows that the Ilimit pulse can be simultaneous with the PWM0 edge. The consequence would be an increased reaction time (132 ns instead of 100 ns).

5. Revision History

Table 3.   Revision history

Revision number	Date	Substantive changes
1	09/2013	Initial release
2	06/2016	*Table 2* – 3 new rows added

Appendix A. Set-up with the MPC5643L Evaluation Board

The attached CodeWarrior project can be used as example for MPC5643L, and loaded on a standard evaluation board. The standard evaluation board, consisting of a generic MPC56xx mother board plus a specific MPC564xL daughter board, is used.

The MPC5643L has to be configured in Lock-Step mode (LSM) for running this project (using Decoupled Parallel mode is possible but requires some software adjustment, such as RAM mapping).

The signals generated by MPC5643L can be observed on the following pins of the evaluation board:

Signal Description	MCU pin	Port Name	PCR or PSMI number	EVB pin (motherboard)
PWM0_H	FlexPWM0_A[0]	A[11]	11	PJ9[12]
PWM0_L	FlexPWM0_B[0]	A[10]	10	PJ9[11]
PWM1_H	FlexPWM0_A[1]	C[7]	39	PJ3[8]
PWM1_L	FlexPWM0_B[1]	C[6]	38	PJ1[8]
Ilimit (fake)	FlexPWM0_A[3]	A[2]	2	PJ9[3]
Analog input for current measurement	ADC1_AN[5]	E[0]	64	PJ5[1] – connected via the jumper J40 to the on-board potentiometer
CTU external trigger	CTU_0[EXT_TRG]	C[14]	46	PJ3[15]
DMA debug pin for force-out configuration	GPIO A[0]	A[0]	0	PJ9[1]
DMA debug pin for current acquisition	GPIO A[3]	A[3]	3	PJ9[4]
eTimer input for Ilimit	eTimer_0 ETC[0]	D[10]	58, PSMI[35], PADSEL=1	PJ8[11]
eTimer output for Ilimit	eTimer_0 ETC[1]	A[1]	1	PJ9[2]
Clock out	GPIO B[6]	B[6]	22	PJ7[7]

To test the software without any power stage, two signals are simulated:

- The output voltage through the load is simulated by the potentiometer W1, connected to analog input channel 5 on ADC1 via jumper J40.
- The Ilimit pulse is generated by FlexPWM0 submodule 4; a pulse is generated every 10 to 15 periods of the PWM signals. It should be connected to the eTimer0 input via a jumper connecting the pins PJ9[3] and PJ8[11].

Document Number: AN4794
Rev. 2
06/2016