Configuring UCD3138 for Full Bridge LLC Converter with Frequency and Phase Shift Modulation Control

Fan Wang *High Performance Isolated Power*

Abstract

Full bridge LLC converter with frequency and phase shift modulation control has been proved to have the ability to support a wide range of operation with improved efficiency or controllability. [1] TI's UCD3138 highly integrated digital controller provides flexible peripherals to support this hybrid control with seamless mode switching. This application notes discusses how to configure UCD3138 for full bridge LLC converter with frequency and phase shift modulation control. Detailed explanation of the relevant peripherals is provided. At the end of this application note, code example is provided.

Contents

1 Introduction

The UCD3138 digital controller has a very powerful digital power control peripheral. The DPWM logic is probably the most complex of the digital peripherals. It takes the output of the compensator and converts it into the correct DPWM output. The DPWM modules have different modes of operation to meet the requirements of different power supply topologies. For details about the different modes of operation, please refer to UCD3138 datasheet. In this full bridge LLC application, Normal Mode, Multiple Output Mode (Multi Mode), and Resonant Mode are used. The DPWM module has hardware for generating complex waveforms – DPWMC, the Edge Generation Module, and the IntraMux. In this application, all these three modules are used. The UCD3138 digital controller supports automatic mode switching, which enables the DPWM module to switch between modes automatically, with no firmware intervention. In this application, all three levels of automatic mode switching are used.

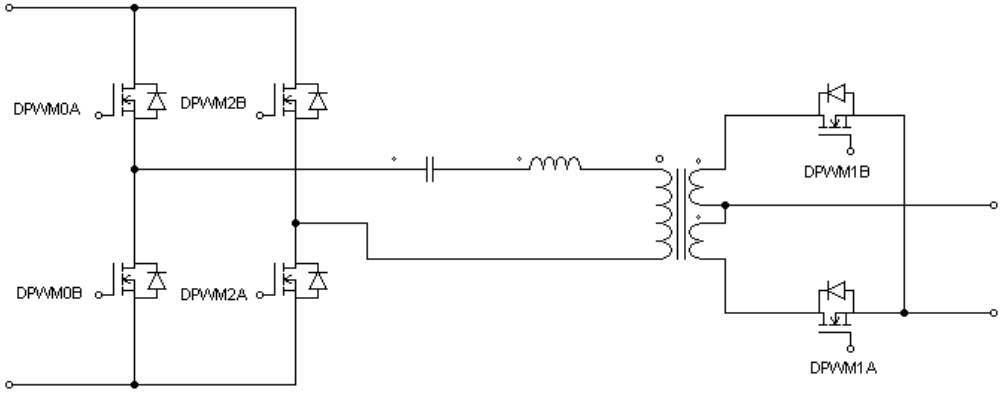
Because of the complexity of the configuration, we suggest you read the UCD3138 datasheet and understand the following, before proceed to read this document in detail:

- Normal Mode (UCD3138 datasheet Section 4.5.1)
- Multi Mode (UCD3138 datasheet Section 4.7)
- Resonant Mode (UCD3138 datasheet Section 4.8)
- Automatic Mode Switching (UCD3138 datasheet Section 4.12)
- DPWMC, Edge Generation, IntraMux (UCD3138 datasheet Section 4.13)

2 Hardware Configuration

The hardware connections described in this document are based on the UCD3138 full bridge LLC reference design.

DPWM connections:



DPWM0A – Primary top left
DPWM0B – Primary bottom left
DPWM2A – Primary bottom right
DPWM2B – Primary top right
DPWM1A – Secondary negative phase
DPWM1B – Secondary positive phase

3 Mode Switching Configuration

3.1 Overview

The LLC with phase shift modulation uses 3 levels of automatic mode switching.
The mode sequence is:

- Mode 1: Light load and/or high input voltage - phase shift modulation for the primary switches, multi-mode for the synchronous rectifiers
- Mode 2: Medium load - frequency modulation for both the primary and secondary switches
- Mode 3: High output current/low input voltage - All switches operate with frequency modulation, however the synchronous rectifier pulse width is clamped to a maximum value based selected by the user.

The mode switching between the three modes involves both the change of the DPWM mode and the IntraMux.

The mode switching mechanism is shown in the figure below.

In Mode 1, the Intra-Mux and PWM mode are configured in DPWMCTRL0 registers;
In Mode 2, the Intra-Mux and PWM mode are configured in DPWMAUTOMID registers;
In Mode 3, the Intra-Mux and PWM mode are configured in DPWMAUTOMAX registers.

The mode switching boundaries are set by the mode switching thresholds shown in the figure.

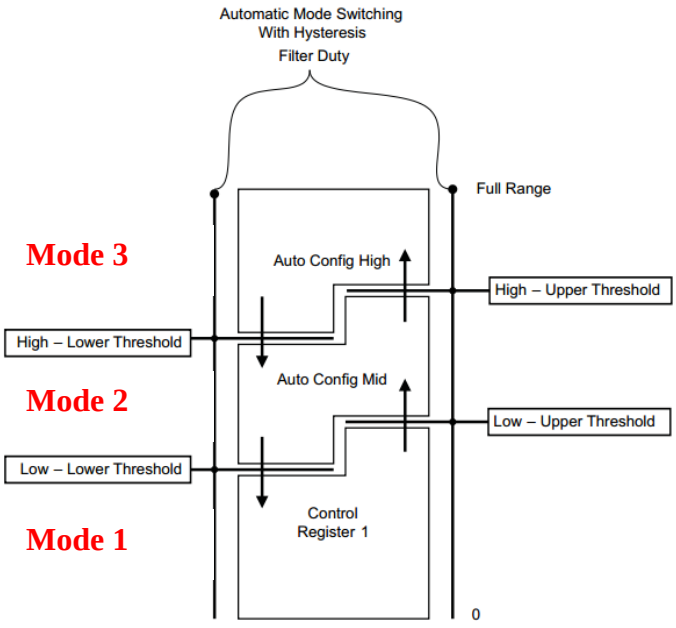


Figure 1 Mode Switching Mechanism

The table below is a summary of the settings used for each mode:

Table 1 Mode Switching Configuration

DPWM module	DPWM pin	Master/ slave	Mode 1 (CTRL0)		Mode 2 (AUTOMID)		Mode 3 (AUTOMAX)	
			INTRAMUX	DPWM MODE	INTRAMUX	DPWM MODE	INRAMUX	DPWM MODE
0	0A	Slave	Below C	Normal	Pass through	Resonant	Pass through	Resonant
	0B		Below 2 C		Pass through		Pass through	
	0C							
1	1A	Slave	Cross over	Multi	Cross over	Resonant	Cross over	Resonant with fixed SR duty
	1B		Cross over		Cross over		Cross over	
	1C							
2	2A	Master	EDGEGEN	Normal	Below A	Resonant	Below A	Resonant
	2B		EDGEGEN		Below B		Below B	
	2C							

In this table, column “DPWM pin” means the physical pin on the IC. The IntraMux selects which signal to appear on the physical pin. For example, in DPWM0 Mode 1, IntraMux is configured as “Below C”, meaning that DPWM1C signal appears on DPWM0A pin; In DPWM0 Mode 1, “Below 2 C” means DPWM2C signal apprears on DPWM0B pin; In DPWM0 Mode 2, “Pass through” means DPWM0A signal appears on DPWM0A pin, DPWM0B signal appears on DPWM0B pin. “Cross over” means DPWMA signal appears on DPWMB and DPWMB signal appears on DPWMA.

For details about multi-mode, resonant mode, and normal mode, please refer to UCD3138 programmer’s manual.

In the next few sections, detailed configurations of the three modes are described.

3.2 Mode 1 – Phase-shift modulation for the primary, Multi-mode for the SRs

In this mode, the DPWM configuration is similar with UCD3138 Phase Shifted Full Bridge DPWM configuration. It makes use of the Intra-Mux and Edge Generation modules of the DPWM modules.

As shown in Figure 2, the top 6 signals in black are signals on the DPWM pin. The comments in the parentheses shows the IntraMux configuration. The bottom 4 signals are internal signals which don’t appear on the pins.

The black dash lines are fixed edges, while the red dash lines are moving edges. 0A and 0B are fixed signals generated from DPWM1C and DPWM2C. 2A and 2B takes edges from internal DPWM2 and DPWM3 to generate 4 moving edges. When the filter output changes, the phase between DPWM0 and DPWM2 will change. DPWM2 and DPWM3 are configured in normal mode.

DPWM1 module is used to drive the secondary side MOSFETs. It is configured in multi mode. The pulse width of 1B and 1A changes with the phase shift angle.

The dead time T1 is configured by EV3-EV2 in DPWM3 and DPWM2 module. The dead time T2 is configured by EV1 and DPWMC event register setting, and cycle adjustment registers.

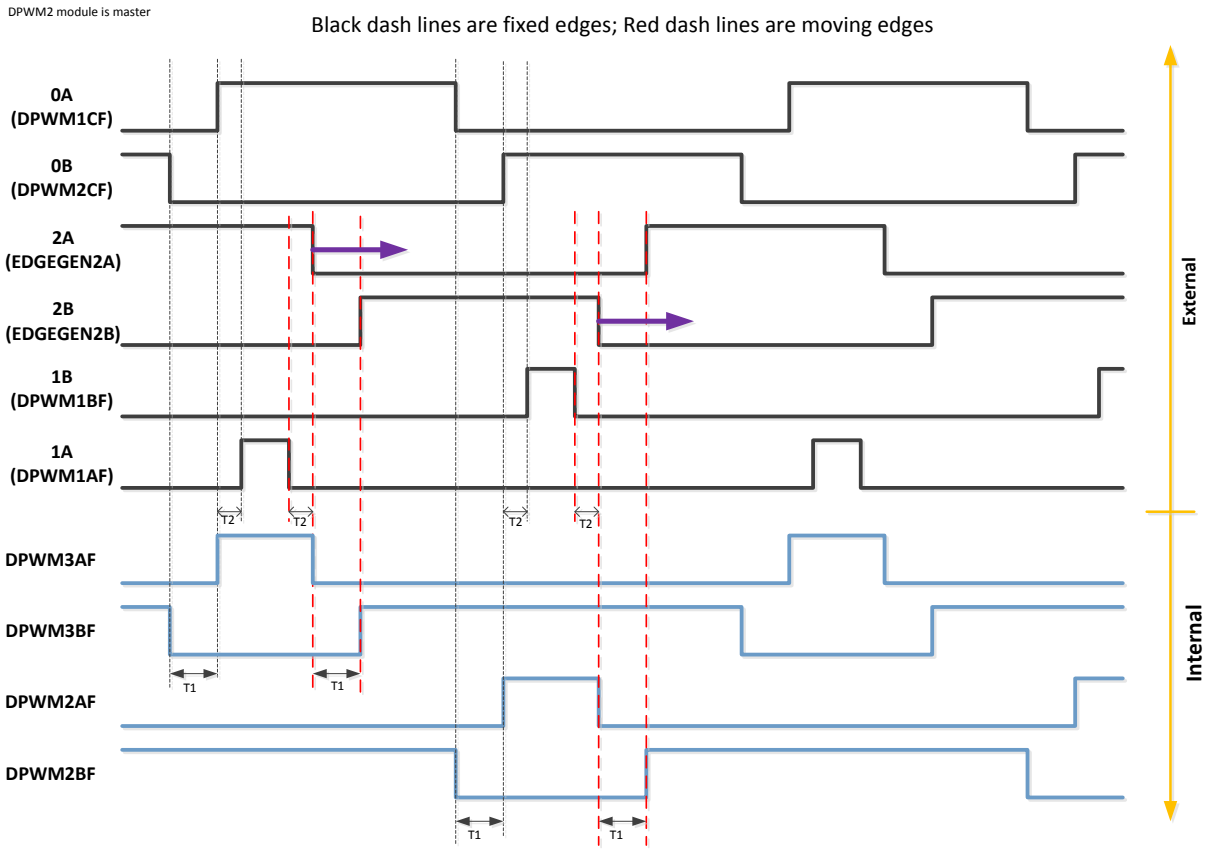


Figure 2 Mode 1 DPWM waveforms

3.2.1 DPWM Synchronization

One of the primary side signals, DPWM2, is configured to be master, and all other DPWM modules are slaved to DPWM2. The phase shift between DPWM modules are set by DPWM phase trigger register.

```
Dpwm0Regs.DPWMCTRL0.bit.MSYNC_SLAVE_EN = 0; //configured to master
Dpwm1Regs.DPWMCTRL0.bit.MSYNC_SLAVE_EN = 1; //configured to slave
Dpwm2Regs.DPWMCTRL0.bit.MSYNC_SLAVE_EN = 1; // configured to slave

LoopMuxRegs.DPWMMUX.bit.DPWM0_SYNC_SEL = 2;     // Slave to dpwm-2
LoopMuxRegs.DPWMMUX.bit.DPWM1_SYNC_SEL = 2;     // Slave to dpwm-2
LoopMuxRegs.DPWMMUX.bit.DPWM3_SYNC_SEL = 2;     // Slave to dpwm-2

Dpwm0Regs.DPWMPHASETRIG.all =  PWM_SLAVESYNC;  //180 degree shift
Dpwm2Regs.DPWMPHASETRIG.all =  PWM_SLAVESYNC;  //180 degree shift
```

Note that DPWM0 (not used in this mode) and DPWM2 are in phase, DPWM1 and DPWM3 are in phase but with a 180 degree phase shift. To make the correct phase shift, either pair should be given a 180 degree phase shift by setting the phase trigger register value. The other pair's phase shift is 0 as default.

3.2.2 Fixed Signals to Bridge – 0A, 0B

The two top signals in the drawing above have fixed timing. The DPWM1CF and DPWM2CF signals are used for these pins. DPWMCxF refers to the signal coming out of the fault module of DPWMx. These signals are actually routed to pins DPWM3A and 3B using the Intra Mux with these statements:

```
Dpwm0Regs.DPWMCTRL0.bit.PWM_A_INTRA_MUX  = 7;   // Send DPWM1C
Dpwm0Regs.DPWMCTRL0.bit.PWM_B_INTRA_MUX  = 8;   // Send DPWM2C
```

Since these signals are really being used as events in the timer, the #defines are called EV5 and EV6. Here are the statements which initialize them:

```
// Setup waveform for DPWM-C (re-using blanking B regs)
Dpwm2Regs.DPWMBLKBBEG.all =  PWM2_EV5;
Dpwm2Regs.DPWMBLKBEND.all =  PWM2_EV6;
```

The statements for DPWM1 are the same. Remember that DPWMC reuses the Blank B registers for timing information.

3.2.3 Dynamic Signals to Bridge – 2A, 2B

The next two signals are the dynamic signals to the bridge. They are generated using the Edge Generator Module in DPWM2. The Edge Generator sources are DPWM2 and DPWM3. The edges used are:

DPWM2A turned on by a rising edge on DPWM2BF
DPWM2A turned off by a falling edge on DPWM3AF
DPWM2B turned on by a rising edge on DPWM3BF
DPWM2B turned off by a falling edge on DPWM2AF

The Edge Generator is configured with these statements:

```
Dpwm2Regs.DPWMEDGEGEN.bit.A_ON_EDGE  =  2;
Dpwm2Regs.DPWMEDGEGEN.bit.A_OFF_EDGE =  5;
Dpwm2Regs.DPWMEDGEGEN.bit.B_ON_EDGE  =  6;
Dpwm2Regs.DPWMEDGEGEN.bit.B_OFF_EDGE =  1;
```

```
Dpwm2Regs.DPWMCTRL0.bit.PWM_A_INTRA_MUX = 1; // output EDGEGEN-A
Dpwm2Regs.DPWMCTRL0.bit.PWM_B_INTRA_MUX = 1; // output EDGEGEN-B

Dpwm2Regs.DPWMEDGEGEN.bit.EDGE_EN = 1; // enable EDGEGEN module
```

Only one EDGE_EN bit is shown here, but the EDGE_EN bits are set for all 4 DPWMs. This is done to ensure that all have signals have the same timing delay through the DPWM.

In Mode 1, DPWM2 and DPWM3 modules are in normal mode. There is a phase delay of 180 degree between DPWM2 and DPWM3. The delay is set by the phase trigger register. In the figure, the black dash lines are fixed edges; the red dash lines are moving edges. Since DPWM2 and DPWM3 are in normal mode, the two red edges will move in the same direction with the same step length as CLA value changes. In this way, the extracted signals on the edge generation module will shift its phase while keeping the duty the same.

```
Dpwm2Regs.DPWMCTRL0.bit.PWM_MODE = 0; // normal mode
Dpwm3Regs.DPWMCTRL0.bit.PWM_MODE = 0; // normal mode
```

3.2.4 SR Configuration

The SRs are driven by the DPWM1A and DPWM1B pins. The internal signals come from DPWM1BF and DPWM1AF respectively (due to hardware routing optimization), and IntraMux is set to “crossover”:

```
Dpwm1Regs.DPWMCTRL0.bit.PWM_A_INTRA_MUX  = 3;   // Send DPWM1BF
Dpwm1Regs.DPWMCTRL0.bit.PWM_B_INTRA_MUX  = 3;   // Send DPWM1AF
```

DPWM1 module is configured to be in Multi-Mode.

```
Dpwm1Regs.DPWMCTRL0.bit.PWM_MODE = 2; // multi mode
```

3.3 Mode 2 – Frequency modulation for both primary and secondary switches

In this mode, the DPWM2A and DPWM2B pins output identical signals with DPWM0A and DPWM0B. All of them use frequency modulation. The synchronous rectifiers also use frequency modulation. The diagram below shows the signals:

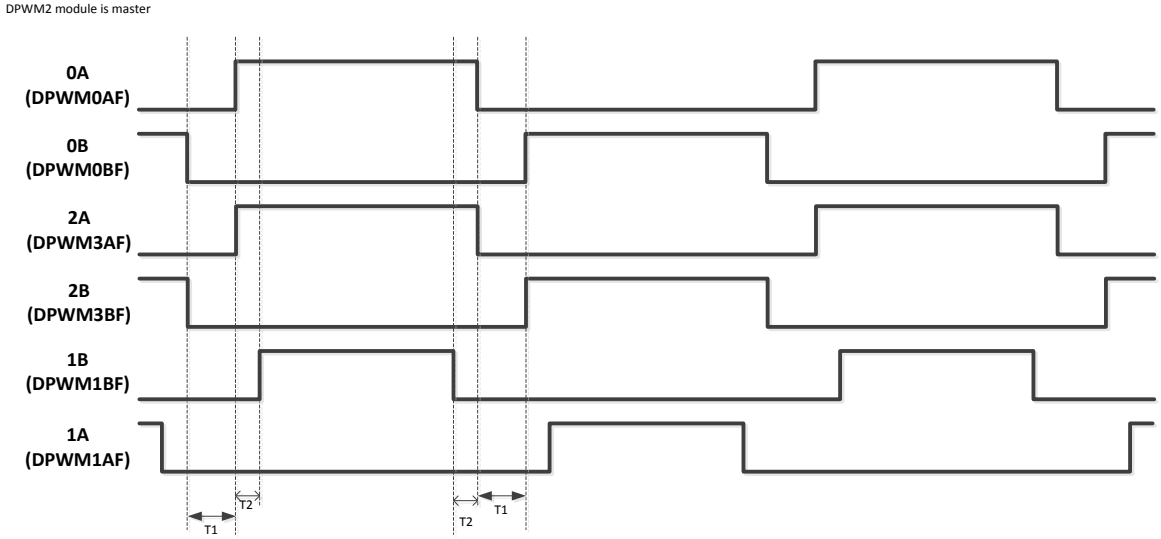


Figure 3 Mode 2 DPWM waveforms

3.3.1 Primary Side DPWMs

DPWM2A, 2B are identical with DPWM0A, 0B in this mode.

```
Dpwm0Regs.DPWMAUTOMID.bit.PWM_A_INTRA_MUX  = 0;   // Send DPWM0AF
Dpwm0Regs.DPWMAUTOMID.bit.PWM_B_INTRA_MUX  = 0;   // Send DPWM0BF
Dpwm2Regs.DPWMAUTOMID.bit.PWM_A_INTRA_MUX  = 4;   // Send DPWM3AF
Dpwm2Regs.DPWMAUTOMID.bit.PWM_B_INTRA_MUX  = 5;   // Send DPWM3BF
```

Note that DPWMs in Mode 2 need to be configured in DPWMAUTOMID registers. DPWM0 and DPWM3 module need to be in resonant mode:

```
Dpwm0Regs.DPWMAUTOMID.bit.PWM_MODE  = 1;  //resonant mode
Dpwm3Regs.DPWMAUTOMID.bit.PWM_MODE  = 1;  //resonant mode
```

3.3.2 SR Configuration

The IntraMux configurations for SRs are the same as in Mode 1. So the same IntraMux configuration can be copied from DPWMCTRL0 registers to DPWMAUTOMID registers. The PWM mode is changed to resonant mode:

```
Dpwm1Regs.DPWMAUTOMID.bit.PWM_MODE = 1; // resonant mode
```

3.4 Mode 3 – Frequency Modulation for both sides, SR fixed pulse width

In this mode, the primary side DPWMs are in the same configuration with in Mode 2, while the SR configuration is resonant mode with fixed pulse width. To enable fixed pulse width on SR, the following bits need to be set in DPWMAUTOMAX registers. Other bits of DPWMAUTOMAX registers can be copied from DPWMAUTOMID registers.

```
Dpwm1Regs.DPWMAUTOMAX.bit.RESON_MODE_FIXED_DUTY  = 1;
```

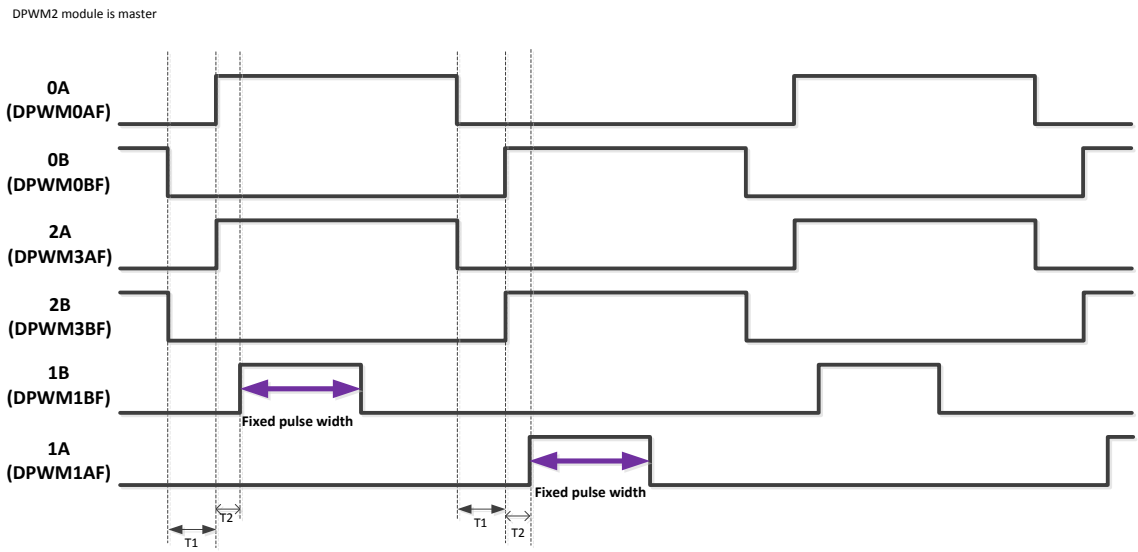


Figure 4 Mode 3 DPWM waveforms

4 DPWM Timing Configuration

In different PWM_MODE, the edges (shown in the figure below: E1, E2, E3, and E4) of DPWM waveforms are controlled by different registers. For the transition between different operation modes smoothly, the timing of edges in different modes should match at the boundary. This chapter talks about the timing configuration for the DPWM modules.

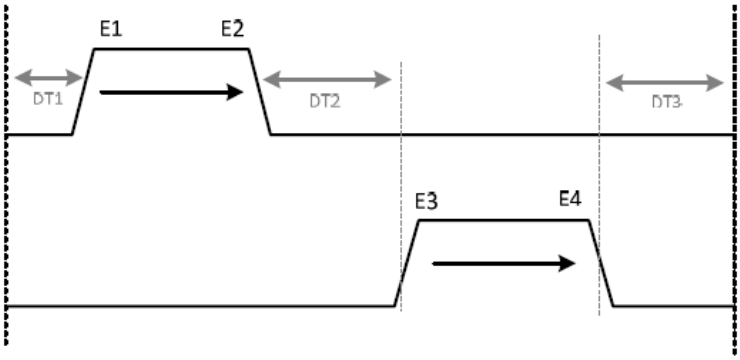


Figure 5 One frame DPWM

4.1 Normal Mode Overview

The following table shows how the edges are controlled in normal mode:

Table 2 DPWM timing in normal mode

Edge/Dead time	Controlled by
E1	EV1
E2	EV1 + filter duty + CYCLE_ADJUST_A
E3	EV1 + filter duty + CYCLE_ADJUST_A – (EV3 – EV2)
E4	EV4
DT1	EV1
DT2	EV3 – EV2
DT3	PRD – EV4

4.2 Multi-Mode Overview

The following table shows how the edges are controlled in multi-mode:

Table 3 DPWM timing in multi-mode

Edge/Dead time	Controlled by
E1	EV1 register
E2	EV1 + filter duty + CYCLE_ADJUST_A
E3	EV3
E4	EV1 + filter duty + CYCLE_ADJUST_B

4.3 Resonant Mode Overview

The following table shows how the edges are controlled in resonant mode:

Table 4 DPWM timing in resonant mode

Edge/Dead time	Controlled by
E1	EV1
E2	EV1 + filter duty + (– DTtotal/2)
E3	EV1 + filter duty + (EV3 – EV2) + (– DTtotal/2)
E4	filter period – (PRD – EV4)
DT1	EV1
DT2	EV3 – EV2
DT3	PRD – EV4

DTtotal = DT1 + DT2 + DT3.

Note that the term (– DTtotal/2) won't be added unless resonant mode dead time compensation is enabled.

```
Dpwm0Regs.DPWMCTRL2.bit.RESON_DEADTIME_COMP_EN = 1;
Dpwm1Regs.DPWMCTRL2.bit.RESON_DEADTIME_COMP_EN = 1;
Dpwm2Regs.DPWMCTRL2.bit.RESON_DEADTIME_COMP_EN = 1;
Dpwm3Regs.DPWMCTRL2.bit.RESON_DEADTIME_COMP_EN = 1;
```

Before the bit is enabled, the pulse width of DPWMxA and DPWMxB will be different:

Pulse width of DPWMxA = E2 – E1 = filter duty;
Pulse width of DPWMxB = E4 – E3 = filter period – filter duty – (PRD–EV4–EV1–EV2–EV3) ;

The term (PRD–EV4–EV1–EV2–EV3) is the total dead time in one switching cycle. Generally, the ratio between filter period and filter duty is fixed in resonant mode: filter duty = ½ * filter period. How to configure the ratio to be ½ will be covered in the next section. Then the pulse width of DPWMxB becomes (filter duty – DTtotal), which is less than DPWMxA pulse width.

Therefore, resonant dead time compensation enable bit should always be set in resonant mode to ensure symmetrical waveforms.

4.4 Filter and LoopMux Configuration

In the above three modes, three different PWM_MODE are used. For the three PWM_MODE to work properly, the output stage of the filter needs to be configured carefully.

The output stage of the filter contains two multiplexors which can select a variety of correction factors to match the DPWM period and DPWM duty.

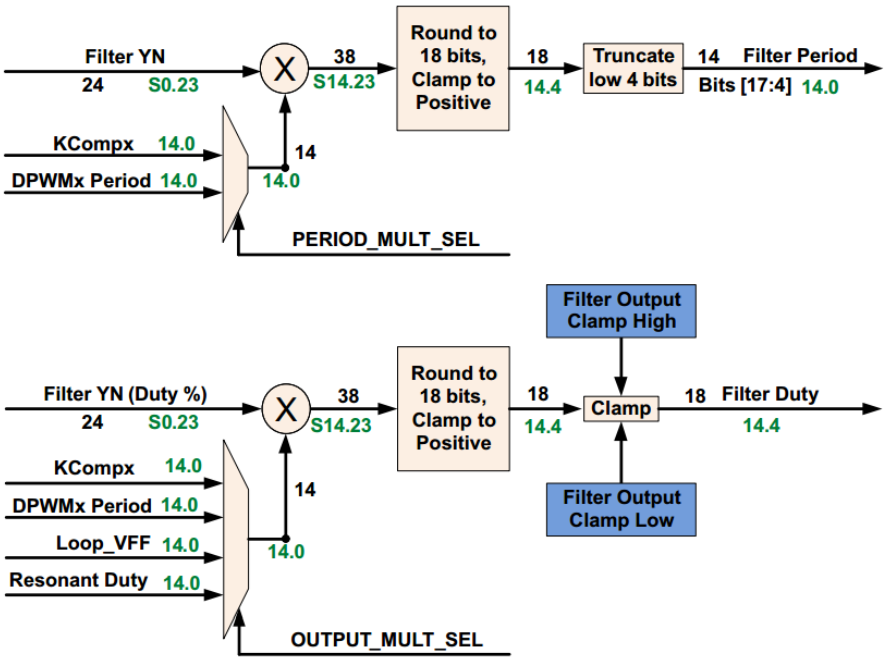


Figure 6 Filter output stage

The detailed structure of filter duty multiplexor is shown in the figure below:

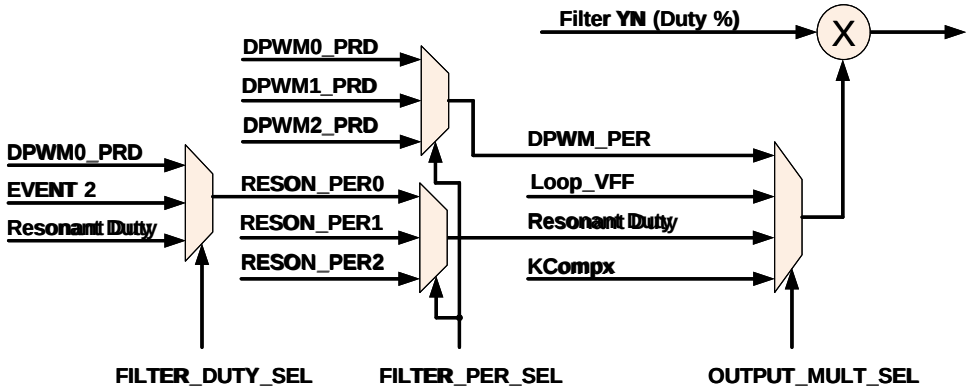


Figure 7 Filter duty multiplexor

4.4.1 KCOMP and Filter Period

The filter period multiplication factor is configured to be KCOMP, which is a fixed value:

```
Filter0Regs.FILTERCTRL.bit.PERIOD_MULT_SEL = 1; // use KCOMP
```

The value of KCOMP is set in the LoopMux register:

```
LoopMuxRegs.FILTERKCOMPA.bit.KCOMP0 = MAXPER;
```

4.4.2 Filter Duty

Filter duty is configured to be YN multiply by resonant duty received from DPWM module:

Filter0Regs.FILTERCTRL.bit.OUTPUT_MULT_SEL = 3; //use resonant duty for scaling

The resonant duty is associated with DPWM3 module:

LoopMuxRegs.FILTERMUX.bit.FILTER0_PER_SEL = 3; //use DPWM3 period

Besides, the register used for maximum duty should be selected to be PWM period adjust register. To keep the DPWM timing configuration simple, 4 DPWM modules use the same setting:

```
Dpwm0Regs.DPWMCTRL2.bit.FILTER_DUTY_SEL = 2;
Dpwm1Regs.DPWMCTRL2.bit.FILTER_DUTY_SEL = 2;
Dpwm2Regs.DPWMCTRL2.bit.FILTER_DUTY_SEL = 2;
Dpwm3Regs.DPWMCTRL2.bit.FILTER_DUTY_SEL = 2;
```

4.5 Timing Matching

4.5.1 Primary Side DPWM Timing Matching

The table below shows how the primary side edges are controlled in each mode:

Table 5 Primary side DPWM timing matching

Edge	Mode 1	Mode 2 or Mode 3
3A on	PWM1EV5	PWM3EV1
3A off	PWM1EV6	PWM3EV1 + filter duty – DTSRtotal/2
3B on	PWM2EV5 + PRD/2	PWM3EV1 + filter duty + (PWM3EV3 – PWM3EV2) – DTSRtotal/2
3B off	PWM2EV6 + PRD/2	filter period – (PRD – PWM3EV4)
2A on	PWM2EV1 + filter duty + CYCLE_ADJUST_A + (PWM2EV3 – PWM2EV2) + PRD/2	Same with 3A on
2A off	PWM3EV1 + filter duty + CYCLE_ADJUST_A	Same with 3A off
2B on	PWM3EV1 + filter duty + CYCLE_ADJUST_A + (PWM3EV3 – PWM3EV2)	Same with 3B on
2B off	PWM2EV1 + filter duty + CYCLE_ADJUST_A + PRD/2	Same with 3B off

At the mode switching boundary, filter duty = PRD/2. For the edges to match, the following settings are needed (In the table, Column 2 = Column 3):

PWM1EV5 = PWM3EV1 = DT1
PWM1EV6 = DT1 + PRD/2 – (DT1+DT2+DT3)/2
PWM2EV5 = DT1 + DT2 – (DT1+DT2+DT3)/2
PWM2EV6 = PWM3EV4 – PRD/2 = PRD – DT3

PWM2EV1 = PWM3EV1 = DT1
PWM2EV2 = PWM3EV2 = PRD/2 – DT2/2
PWM2EV3 = PWM3EV3 = PRD/2 + DT2/2
PWM2EV4 = PWM3EV4 = PRD – DT3

CYCLE_ADJUST_A = – (PWM2EV3 – PWM2EV2) = – DT2

4.5.2 SR DPWM Timing Matching

The table below shows how the SR edges are controlled in each mode:

Table 6 Secondary side DPWM timing matching

Edge	Mode 1	Mode 2 or Mode 3
1A on	PWM1EV1	PWM1EV1
1A off	PWM1EV1 + filter duty + CYCLE_ADJUST_A	PWM1EV1 + filter duty – DTtotal/2 or fixed duty
1B on	PWM1EV3	PWM1EV1 + filter duty – DTtotal/2 + (PWM1EV2 – PEM1EV3)
1B off	PWM1EV3 + filter duty + CYCLE_ADJUST_B	filter period – (PRD – PWM1EV4) or fixed duty

Make Column 2 = Column 3, we get the following:

CYCLE_ADJUST_A = CYCLE_ADJUST_B = – DTtotal/2 = – DTSR2
PWM1EV1 = DTSR1
PWM1EV2 = PRD/2 – DTSR2/2
PWM1 EV3 = PRD/2 + DTSR2/2
PWM1EV4 = PRD – DTSR3

4.6 Mode Switching Thresholds

The mode switching threshold between Mode 1 and Mode 2 is ½ of the desired period to switch modes; the mode switching threshold between Mode 2 and Mode 3 is equal to the period at the resonant point.

5 Other Considerations (Important)

Note that because of the use of cycle adjustment registers, clamps should be set in order to prevent negative filter duty value. In this application, two levels of clamps are set for secondary side DPWMs and primary side DPWMs respectively.

5.1 Minimum Duty Setting for Secondary Side DPWMs

Referring to Figure 2, the first level of clamp is set on DPWM1 to prevent it from getting negative duty. It is set by DPWM1 minimum duty setting register.

```
Dpwm1Regs.DPWMMINDUTYLO.bit.MIN_DUTY_LOW = 263;
Dpwm1Regs.DPWMMINDUTYHI.bit.MIN_DUTY_HIGH = 263;
```

5.1 Filter Output Clamp

The second clamp is to prevent DPWM2 and DPWM3 gets negative duty. It is set by filter output clamp low register. Below are the #defines for filter output clamp low registers:

```
#define INIT_FILTER_0_OUTPUT_CLAMP_LOW  (1600)
#define INIT_FILTER_0_START_UP_OUTPUT_CLAMP_LOW  (1600)
#define INIT_FILTER_0_CP_OUTPUT_CLAMP_LOW  (1600)
#define INIT_FILTER_1_OUTPUT_CLAMP_LOW  (1600)
```

5.2 Burst Mode

As stated before, the filter controlled phase shift is created by DPWM2 and DPWM3 edges. Because EV2 cannot go before EV1, the phase shift angle cannot go 180 degrees, considering the delays in the circuit and the dead time requirements. Thus there is still energy transfer to the secondary side even if the duty of DPWM2 and DPWM3 becomes zero. To solve this problem, hardware burst mode is used when the filter output is very small.

Below are the settings for this application:

```
Dpwm0Regs.DPWMCTRL1.bit.BURST_EN = 1;
Dpwm1Regs.DPWMCTRL1.bit.BURST_EN = 1;
Dpwm2Regs.DPWMCTRL1.bit.BURST_EN = 1;
Dpwm3Regs.DPWMCTRL1.bit.BURST_EN = 1;

LoopMuxRegs.LLCTRL.bit.LL_FILTER_SEL = 0;
LoopMuxRegs.LLENTHRESH.bit.TURN_ON_THRESH = 3300;
LoopMuxRegs.LLDISTHRESH.bit.TURN_OFF_THRESH = 3300;
```

5.3 Faults

When fault happens and DPWMs need to turn off, it is required to turn off DPWMs by changing them to GPIO mode and set the output to 0.

6 References

1. B. McDonald, F. Wang, “Digital Control Provides LLC Performance Enhancements”, Power Electronics Technologies, August 2013
2. UCD3138 Highly Integrated Digital Controller for Isolated Power datasheet

7 Code Example

```
//High resolution DPWM step size
#define  TDPWM            (250e-12)
//EADC sample rate
#define  TSAMP            (1 / (100e3))

#define  MINPER           ((int)(TH / TDPWM / 16 + 0.5))
#define  MAXPER           ((int)(TL / TDPWM / 16 + 0.5))

//PWM mode period. This is also the maximum resonant frequency
#define  TH               (1 / (85e3))
//Lowest resonant mode period.
#define  TL               (1 / (35e3))
//Primary MOSFET dead time
#define  DT0              (350e-9)
```

```
//SR dead time
#define  DT0SR            (1050e-9)
//Maximum SR on time
#define  TSR              (6.2e-6 + DT0SR)

//Primary switches dead time from start of period to EV1
#define  DT1              (DT0 / 2)
//Primary switches dead time from EV2 to EV3
#define  DT2              (DT0)
//Primary switches dead time from EV4 to end of period
#define  DT3              (DT0 / 2)
//SR switches dead time from start of period to EV1
#define  DT1SR            (DT0SR / 2)
//SR switches dead time from EV2 to EV3
#define  DT2SR            (DT0SR)
//SR switches dead time from EV4 to end of period
#define  DT3SR            (DT0SR / 2)
//Primary switches total dead time
#define  DT               (DT1 + DT2 + DT3)
//SR switches total dead time
#define  DTSR             (DT1SR + DT2SR + DT3SR)

#define  TSRMAX           ((int)((TSR - DT2SR) / TDPWM / 16 + 0.5))
#define  DEADTIME1        ((int)(DT1 / TDPWM + 0.5))
#define  DEADTIME2        ((int)(DT2 / TDPWM + 0.5))
#define  DEADTIME3        ((int)(DT1SR / TDPWM + 0.5))
#define  DEADTIME4        ((int)(DT2SR / TDPWM + 0.5))

#define  HALF_PERIOD      ((MINPER + 1) >> 1)

  // ---------------------- DPWM0
#define  PWM0_EV1         ((DEADTIME3 + 8) >> 4)
#define  PWM0_EV2         (MINPER * 8 - (DEADTIME4 >> 1))
#define  PWM0_EV3         (MINPER * 8 + (DEADTIME4 >> 1))
#define  PWM0_EV4         (MINPER * 16 - DEADTIME3)
#define  PWM0_CYC_ADJ     (-DEADTIME4)
  // ---------------------- DPWM1
#define  PWM1_EV1          ((DEADTIME3 + 8) >> 4)
#define  PWM1_EV2          (MINPER * 8 - (DEADTIME4 >> 1))
#define  PWM1_EV3          (MINPER * 8 + (DEADTIME4 >> 1))
#define  PWM1_EV4          (MINPER * 16 - DEADTIME3)
#define  PWM1_EV5          ((DEADTIME3 + 8) >> 1)
#define  PWM1_EV6          (MINPER * 8 - (DEADTIME4 >> 2))
#define  PWM1_CYC_ADJ      (-DEADTIME4)
  // ---------------------- DPWM2
#define  PWM2_EV1          ((DEADTIME1 + 8) >> 4)
#define  PWM2_EV2          (MINPER * 8 - (DEADTIME2 >> 1))
#define  PWM2_EV3          (MINPER * 8 + (DEADTIME2 >> 1))
#define  PWM2_EV4          (MINPER * 16 - DEADTIME1)
#define  PWM2_EV5          ((DEADTIME3 + 8) >> 1)
#define  PWM2_EV6          (MINPER * 8 - (DEADTIME4 >> 2))
#define  PWM2_CYC_ADJ      (-DEADTIME2)
  // ---------------------- DPWM3
#define  PWM3_EV1          ((DEADTIME1 + 8) >> 4)
#define  PWM3_EV2          (MINPER * 8 - (DEADTIME2 >> 1))
#define  PWM3_EV3          (MINPER * 8 + (DEADTIME2 >> 1))
#define  PWM3_EV4          (MINPER * 16 - DEADTIME1)
#define  PWM3_CYC_ADJ      (-DEADTIME2)

  // ----------------------Intra-Mux
#define  PASS_THROUGH   (0)
#define  EDGEGEN        (1)
#define  PWMC           (2)
```

```
#define  CROSSOVER      (3)
#define  BELOW_A        (4)
#define  BELOW_B        (5)
#define  BELOW_C        (6)
#define  BELOW2_C       (7)
#define  BELOW3_C       (8)
  // ----------------------Edge-Gen
#define  CURRENT_POS_A  (0)
#define  CURRENT_NEG_A  (1)
#define  CURRENT_POS_B  (2)
#define  CURRENT_NEG_B  (3)
#define  NEXT_POS_A     (4)
#define  NEXT_NEG_A     (5)
#define  NEXT_POS_B     (6)
#define  NEXT_NEG_B     (7)
  // ----------------------PWM modes
#define  NORMAL         (0)
#define  RESONANT       (1)
#define  MULTI          (2)

void init_dpwm0(void) // DPWM0 is used for primary side drive.
{
    // --------------- CTRL0
    Dpwm0Regs.DPWMCTRL0.bit.PWM_EN = 1;
    Dpwm0Regs.DPWMCTRL0.bit.CLA_EN = 1;
    Dpwm0Regs.DPWMCTRL0.bit.PWM_MODE = NORMAL;
    // Slave
    Dpwm0Regs.DPWMCTRL0.bit.MSYNC_SLAVE_EN = 1;
    // Adv cnt limit enabled
    Dpwm0Regs.DPWMCTRL0.bit.CBC_ADV_CNT_EN = 1;
    // Current limit enabled for AB outputs
    Dpwm0Regs.DPWMCTRL0.bit.CBC_PWM_AB_EN = 1;
    Dpwm0Regs.DPWMCTRL0.bit.PWM_A_INTRA_MUX = BELOW_C;
    Dpwm0Regs.DPWMCTRL0.bit.PWM_B_INTRA_MUX = BELOW2_C;
    Dpwm0Regs.DPWMCTRL0.bit.BLANK_A_EN = 1;
    Dpwm0Regs.DPWMCTRL0.bit.RESON_MODE_FIXED_DUTY_EN = 0;

    // --------------- AUTOMID
    Dpwm0Regs.DPWMAUTOMID.bit.CLA_EN = 1;
    Dpwm0Regs.DPWMAUTOMID.bit.PWM_MODE = RESONANT;
    // Adv cnt limit enabled
    Dpwm0Regs.DPWMAUTOMID.bit.CBC_ADV_CNT_EN = 1;
    // Current limit enabled for AB outputs
    Dpwm0Regs.DPWMAUTOMID.bit.CBC_PWM_AB_EN = 1;
       Dpwm0Regs.DPWMAUTOMID.bit.PWM_A_INTRA_MUX = PASS_THROUGH;
    Dpwm0Regs.DPWMAUTOMID.bit.PWM_B_INTRA_MUX = PASS_THROUGH;
    Dpwm0Regs.DPWMAUTOMID.bit.RESON_MODE_FIXED_DUTY_EN = 0;

    // --------------- AUTOMAX
    Dpwm0Regs.DPWMAUTOMAX.bit.CLA_EN = 1;
    Dpwm0Regs.DPWMAUTOMAX.bit.PWM_MODE = RESONANT;
    // Adv cnt limit enabled
    Dpwm0Regs.DPWMAUTOMAX.bit.CBC_ADV_CNT_EN = 1;
    // Current limit enabled for AB outputs
    Dpwm0Regs.DPWMAUTOMAX.bit.CBC_PWM_AB_EN = 1;
       Dpwm0Regs.DPWMAUTOMAX.bit.PWM_A_INTRA_MUX = PASS_THROUGH;
    Dpwm0Regs.DPWMAUTOMAX.bit.PWM_B_INTRA_MUX = PASS_THROUGH;
    // Enable fixed duty for SR
    Dpwm0Regs.DPWMAUTOMAX.bit.RESON_MODE_FIXED_DUTY_EN = 0;

    // ---------------- CTRL1
    Dpwm0Regs.DPWMCTRL1.bit.HIRES_DIS = 1;
    Dpwm0Regs.DPWMCTRL1.bit.ALL_PHASE_CLK_ENA = 1;
```

```
    Dpwm0Regs.DPWMCTRL1.bit.CHECK_OVERRIDE = 1;
    // Update end of period
    Dpwm0Regs.DPWMCTRL1.bit.EVENT_UP_SEL = 1;
    // Enable auto mode switching
    Dpwm0Regs.DPWMCTRL1.bit.AUTO_MODE_SEL = 1;


    //---------------------------
    // DPWM setting for interrupt
    //---------------------------
    //every 4 cycles
    Dpwm0Regs.DPWMINT.bit.PRD_INT_SCALE =2;
    Dpwm0Regs.DPWMINT.bit.PRD_INT_EN = 1;

}

void init_dpwm1(void) //dpwm1 is used for sync FET
{
    // ------------------ CTRL0
    Dpwm1Regs.DPWMCTRL0.bit.PWM_EN = 1;
    Dpwm1Regs.DPWMCTRL0.bit.CLA_EN = 1;
    Dpwm1Regs.DPWMCTRL0.bit.PWM_MODE = MULTI;
    // Slave
    Dpwm1Regs.DPWMCTRL0.bit.MSYNC_SLAVE_EN = 1;
    // Adv cnt limit enabled
    Dpwm1Regs.DPWMCTRL0.bit.CBC_ADV_CNT_EN = 1;
    Dpwm1Regs.DPWMCTRL0.bit.CBC_PWM_AB_EN = 1;
    // Output 2A, for test only
    Dpwm1Regs.DPWMCTRL0.bit.PWM_A_INTRA_MUX = CROSSOVER;
    Dpwm1Regs.DPWMCTRL0.bit.PWM_B_INTRA_MUX = CROSSOVER;
    Dpwm1Regs.DPWMCTRL0.bit.BLANK_A_EN = 1;
    Dpwm1Regs.DPWMCTRL0.bit.RESON_MODE_FIXED_DUTY_EN = 0;

    // ------------------- AUTOMID
    Dpwm1Regs.DPWMAUTOMID.bit.CLA_EN = 1;
    Dpwm1Regs.DPWMAUTOMID.bit.PWM_MODE = RESONANT;
    // Adv cnt limit enabled
    Dpwm1Regs.DPWMAUTOMID.bit.CBC_ADV_CNT_EN = 1;
    // Current limit enabled for AB outputs
    Dpwm1Regs.DPWMAUTOMID.bit.CBC_PWM_AB_EN = 1;
    // Output 2A, for test only
    Dpwm1Regs.DPWMAUTOMID.bit.PWM_A_INTRA_MUX = CROSSOVER;
    Dpwm1Regs.DPWMAUTOMID.bit.PWM_B_INTRA_MUX = CROSSOVER;
    Dpwm1Regs.DPWMAUTOMID.bit.RESON_MODE_FIXED_DUTY_EN = 0;

    // ------------------- AUTOMAX
    Dpwm1Regs.DPWMAUTOMAX.bit.CLA_EN = 1;
    Dpwm1Regs.DPWMAUTOMAX.bit.PWM_MODE = RESONANT;
    // Adv cnt limit enabled
    Dpwm1Regs.DPWMAUTOMAX.bit.CBC_ADV_CNT_EN = 1;
    // Current limit enabled for AB outputs
    Dpwm1Regs.DPWMAUTOMAX.bit.CBC_PWM_AB_EN = 1;
    // Output 2A, for test only
    Dpwm1Regs.DPWMAUTOMAX.bit.PWM_A_INTRA_MUX = CROSSOVER;
    Dpwm1Regs.DPWMAUTOMAX.bit.PWM_B_INTRA_MUX = CROSSOVER;
    // Enable fixed duty for SR
    Dpwm1Regs.DPWMAUTOMAX.bit.RESON_MODE_FIXED_DUTY_EN = 1;

    // ------------------- CTRL1
    Dpwm1Regs.DPWMCTRL1.bit.HIRES_DIS = 1;
    Dpwm1Regs.DPWMCTRL1.bit.ALL_PHASE_CLK_ENA = 1;
    Dpwm1Regs.DPWMCTRL1.bit.CHECK_OVERRIDE = 1;
    // Update at end of period
```

```
    Dpwm1Regs.DPWMCTRL1.bit.EVENT_UP_SEL = 1;
    // Enable auto mode switching
    Dpwm1Regs.DPWMCTRL1.bit.AUTO_MODE_SEL = 1;
}


void init_dpwm2(void)
{
    // --------------- CTRL0
    Dpwm2Regs.DPWMCTRL0.bit.PWM_EN = 1;
    Dpwm2Regs.DPWMCTRL0.bit.CLA_EN = 1;
    Dpwm2Regs.DPWMCTRL0.bit.PWM_MODE = NORMAL;
    // Master
    Dpwm2Regs.DPWMCTRL0.bit.MSYNC_SLAVE_EN = 0;
    // Adv cnt limit enabled
    Dpwm2Regs.DPWMCTRL0.bit.CBC_ADV_CNT_EN = 1;
    // Current limit enabled for AB outputs
    Dpwm2Regs.DPWMCTRL0.bit.CBC_PWM_AB_EN  = 1;
    Dpwm2Regs.DPWMCTRL0.bit.PWM_A_INTRA_MUX = EDGEGEN;
    Dpwm2Regs.DPWMCTRL0.bit.PWM_B_INTRA_MUX = EDGEGEN;
    Dpwm2Regs.DPWMCTRL0.bit.BLANK_A_EN = 1;

    // ---------------- AUTOMID
    Dpwm2Regs.DPWMAUTOMID.bit.CLA_EN = 1;
    Dpwm2Regs.DPWMAUTOMID.bit.PWM_MODE = RESONANT;
    // Adv cnt limit enabled
    Dpwm2Regs.DPWMAUTOMID.bit.CBC_ADV_CNT_EN = 1;
    // Current limit enabled for AB outputs
    Dpwm2Regs.DPWMAUTOMID.bit.CBC_PWM_AB_EN = 1;
    Dpwm2Regs.DPWMAUTOMID.bit.PWM_A_INTRA_MUX = BELOW_A;
    Dpwm2Regs.DPWMAUTOMID.bit.PWM_B_INTRA_MUX = BELOW_B;

    // ---------------- AUTOMAX
    //make mid and max the same
    Dpwm2Regs.DPWMAUTOMAX.bit.CLA_EN = 1;
    Dpwm2Regs.DPWMAUTOMAX.bit.PWM_MODE = RESONANT;
    // Adv cnt limit enabled
    Dpwm2Regs.DPWMAUTOMAX.bit.CBC_ADV_CNT_EN = 1;
    // Current limit enabled for AB outputs
    Dpwm2Regs.DPWMAUTOMAX.bit.CBC_PWM_AB_EN = 1;
    Dpwm2Regs.DPWMAUTOMAX.bit.PWM_A_INTRA_MUX = BELOW_A;
    Dpwm2Regs.DPWMAUTOMAX.bit.PWM_B_INTRA_MUX = BELOW_B;

    // ---------------- CTRL1
    Dpwm2Regs.DPWMCTRL1.bit.HIRES_DIS = 1;
    Dpwm2Regs.DPWMCTRL1.bit.ALL_PHASE_CLK_ENA = 1;
    Dpwm2Regs.DPWMCTRL1.bit.CHECK_OVERRIDE = 1;
    // Update end of period
    Dpwm2Regs.DPWMCTRL1.bit.EVENT_UP_SEL = 1;
    //enable auto mode switching
    Dpwm2Regs.DPWMCTRL1.bit.AUTO_MODE_SEL = 1;

    // ---------------- Edge-generation setup
    Dpwm2Regs.DPWMEDGEGEN.bit.A_ON_EDGE = CURRENT_POS_B;
    Dpwm2Regs.DPWMEDGEGEN.bit.A_OFF_EDGE = NEXT_NEG_A;
    Dpwm2Regs.DPWMEDGEGEN.bit.B_ON_EDGE = NEXT_POS_B;
    Dpwm2Regs.DPWMEDGEGEN.bit.B_OFF_EDGE = CURRENT_NEG_A;

    // When using edgegen, may need to enable all other DPWMS
    // to ensure same path delay, default is pass thru on others (1234)
    Dpwm0Regs.DPWMEDGEGEN.bit.EDGE_EN = 1;
    Dpwm1Regs.DPWMEDGEGEN.bit.EDGE_EN = 1;
```

```
    Dpwm2Regs.DPWMEDGEGEN.bit.EDGE_EN = 1;
    Dpwm3Regs.DPWMEDGEGEN.bit.EDGE_EN = 1;
}



//DPWM3 is used internally
void init_dpwm3(void)
{
    // --------------- CTRL0
    Dpwm3Regs.DPWMCTRL0.bit.PWM_EN = 1;
    Dpwm3Regs.DPWMCTRL0.bit.CLA_EN = 1;
    Dpwm3Regs.DPWMCTRL0.bit.PWM_MODE = NORMAL;
    // Slave
    Dpwm3Regs.DPWMCTRL0.bit.MSYNC_SLAVE_EN = 1;
    // Adv cnt limit enabled
    Dpwm3Regs.DPWMCTRL0.bit.CBC_ADV_CNT_EN = 1;
    // Current limit enabled for AB outputs
    Dpwm3Regs.DPWMCTRL0.bit.CBC_PWM_AB_EN = 1;
    Dpwm3Regs.DPWMCTRL0.bit.PWM_A_INTRA_MUX = BELOW2_C;
    Dpwm3Regs.DPWMCTRL0.bit.PWM_B_INTRA_MUX = BELOW3_C;
    Dpwm3Regs.DPWMCTRL0.bit.BLANK_A_EN = 1;

    // --------------- AUTOMID
    Dpwm3Regs.DPWMAUTOMID.bit.CLA_EN = 1;
    Dpwm3Regs.DPWMAUTOMID.bit.PWM_MODE = RESONANT;
    // Adv cnt limit enabled
    Dpwm3Regs.DPWMAUTOMID.bit.CBC_ADV_CNT_EN = 1;
    // Current limit enabled for AB outputs
    Dpwm3Regs.DPWMAUTOMID.bit.CBC_PWM_AB_EN = 1;
    Dpwm3Regs.DPWMAUTOMID.bit.PWM_A_INTRA_MUX = PASS_THROUGH;
    Dpwm3Regs.DPWMAUTOMID.bit.PWM_B_INTRA_MUX = PASS_THROUGH;

    // --------------- AUTOMAX
    // make mid and max the same
    Dpwm3Regs.DPWMAUTOMAX.bit.CLA_EN = 1;
    Dpwm3Regs.DPWMAUTOMAX.bit.PWM_MODE = RESONANT;
    // Adv cnt limit enabled
    Dpwm3Regs.DPWMAUTOMAX.bit.CBC_ADV_CNT_EN = 1;
    // Current limit enabled for AB outputs
    Dpwm3Regs.DPWMAUTOMAX.bit.CBC_PWM_AB_EN = 1;
    Dpwm3Regs.DPWMAUTOMAX.bit.PWM_A_INTRA_MUX = PASS_THROUGH;
    Dpwm3Regs.DPWMAUTOMAX.bit.PWM_B_INTRA_MUX = PASS_THROUGH;

    // --------------- CTRL1
    Dpwm3Regs.DPWMCTRL1.bit.HIRES_DIS = 1;
    Dpwm3Regs.DPWMCTRL1.bit.ALL_PHASE_CLK_ENA = 1;
    Dpwm3Regs.DPWMCTRL1.bit.CHECK_OVERRIDE = 1;
    // Update at end of period
    Dpwm3Regs.DPWMCTRL1.bit.EVENT_UP_SEL = 1;
    // enable auto mode switching
    Dpwm3Regs.DPWMCTRL1.bit.AUTO_MODE_SEL = 1;
}

void init_dpwms(void)
{
    init_dpwm0();
    init_dpwm1();
    init_dpwm2();
    init_dpwm3();
    configure_dpwm_timing();
}
```

```
void configure_dpwm_timing(void) // need to change to pmbus variables later
{
    //--------------------
    // Config DPWM timing
    //--------------------
    Dpwm0Regs.DPWMEV1.bit.EVENT1 = PWM0_EV1;
    Dpwm0Regs.DPWMEV2.bit.EVENT2 = PWM0_EV2;
    Dpwm0Regs.DPWMEV3.bit.EVENT3 = PWM0_EV3;
    Dpwm0Regs.DPWMEV4.bit.EVENT4 = PWM0_EV4;
    Dpwm0Regs.DPWMCYCADJA.bit.CYCLE_ADJUST_A = (int16)(-DEADTIME4);
    Dpwm0Regs.DPWMCYCADJB.bit.CYCLE_ADJUST_B = (int16)(-DEADTIME4);
    Dpwm0Regs.DPWMPRD.bit.PRD = MINPER;

    Dpwm1Regs.DPWMEV1.bit.EVENT1 = PWM1_EV1;
    Dpwm1Regs.DPWMEV2.bit.EVENT2 = PWM1_EV2;
    Dpwm1Regs.DPWMEV3.bit.EVENT3 = PWM1_EV3;
    Dpwm1Regs.DPWMEV4.bit.EVENT4 = PWM1_EV4;
    Dpwm1Regs.DPWMBLKBBEG.all = PWM1_EV5;
    Dpwm1Regs.DPWMBLKBEND.all = PWM1_EV6;
    Dpwm1Regs.DPWMCYCADJA.bit.CYCLE_ADJUST_A = (int16)(-DEADTIME4);
    Dpwm1Regs.DPWMCYCADJB.bit.CYCLE_ADJUST_B = (int16)(-DEADTIME4);
    Dpwm1Regs.DPWMPRD.bit.PRD = MINPER;

    Dpwm2Regs.DPWMEV1.bit.EVENT1 = PWM2_EV1;
    Dpwm2Regs.DPWMEV2.bit.EVENT2 = PWM2_EV2;
    Dpwm2Regs.DPWMEV3.bit.EVENT3 = PWM2_EV3;
    Dpwm2Regs.DPWMEV4.bit.EVENT4 = PWM2_EV4;
    Dpwm2Regs.DPWMBLKBBEG.all = PWM2_EV5;
    Dpwm2Regs.DPWMBLKBEND.all = PWM2_EV6;
    Dpwm2Regs.DPWMCYCADJA.bit.CYCLE_ADJUST_A = (int16)(-DEADTIME2);
    Dpwm2Regs.DPWMCYCADJB.bit.CYCLE_ADJUST_B = (int16)(-DEADTIME2);
    Dpwm2Regs.DPWMPRD.bit.PRD = MINPER;

    Dpwm3Regs.DPWMEV1.bit.EVENT1 = PWM3_EV1;
    Dpwm3Regs.DPWMEV2.bit.EVENT2 = PWM3_EV2;
    Dpwm3Regs.DPWMEV3.bit.EVENT3 = PWM3_EV3;
    Dpwm3Regs.DPWMEV4.bit.EVENT4 = PWM3_EV4;
    Dpwm3Regs.DPWMCYCADJA.bit.CYCLE_ADJUST_A = (int16)(-DEADTIME2);
    Dpwm3Regs.DPWMCYCADJB.bit.CYCLE_ADJUST_B = (int16)(-DEADTIME2);
    Dpwm3Regs.DPWMPRD.bit.PRD = MINPER;

    //-----------------------
    // Config Mode Switching
    //-----------------------
    Dpwm0Regs.DPWMAUTOSWHIUPTHRESH.bit.AUTO_SWITCH_HIGH_UPPER = TSRMAX + ((DEADTIME3 +
8) >> 4);
    Dpwm0Regs.DPWMAUTOSWHILOWTHRESH.bit.AUTO_SWITCH_HIGH_LOWER = TSRMAX + ((DEADTIME3
+ 8) >> 4);
    Dpwm0Regs.DPWMAUTOSWLOUPTHRESH.bit.AUTO_SWITCH_LOW_UPPER = HALF_PERIOD;
    Dpwm0Regs.DPWMAUTOSWLOLOWTHRESH.bit.AUTO_SWITCH_LOW_LOWER = HALF_PERIOD;

    Dpwm1Regs.DPWMAUTOSWHIUPTHRESH.bit.AUTO_SWITCH_HIGH_UPPER = TSRMAX + ((DEADTIME3 +
8) >> 4);
    Dpwm1Regs.DPWMAUTOSWHILOWTHRESH.bit.AUTO_SWITCH_HIGH_LOWER = TSRMAX + ((DEADTIME3
+ 8) >> 4);
    Dpwm1Regs.DPWMAUTOSWLOUPTHRESH.bit.AUTO_SWITCH_LOW_UPPER = HALF_PERIOD;
    Dpwm1Regs.DPWMAUTOSWLOLOWTHRESH.bit.AUTO_SWITCH_LOW_LOWER = HALF_PERIOD;

    Dpwm2Regs.DPWMAUTOSWHIUPTHRESH.bit.AUTO_SWITCH_HIGH_UPPER = TSRMAX + ((DEADTIME3 +
8) >> 4);
    Dpwm2Regs.DPWMAUTOSWHILOWTHRESH.bit.AUTO_SWITCH_HIGH_LOWER = TSRMAX + ((DEADTIME3
+ 8) >> 4);
```

```
    Dpwm2Regs.DPWMAUTOSWLOUPTHRESH.bit.AUTO_SWITCH_LOW_UPPER = HALF_PERIOD;
    Dpwm2Regs.DPWMAUTOSWLOLOWTHRESH.bit.AUTO_SWITCH_LOW_LOWER = HALF_PERIOD;

    Dpwm3Regs.DPWMAUTOSWHIUPTHRESH.bit.AUTO_SWITCH_HIGH_UPPER = TSRMAX + ((DEADTIME3 +
8) >> 4);
    Dpwm3Regs.DPWMAUTOSWHILOWTHRESH.bit.AUTO_SWITCH_HIGH_LOWER = TSRMAX + ((DEADTIME3
+ 8) >> 4);
      Dpwm3Regs.DPWMAUTOSWLOUPTHRESH.bit.AUTO_SWITCH_LOW_UPPER = HALF_PERIOD;
      Dpwm3Regs.DPWMAUTOSWLOLOWTHRESH.bit.AUTO_SWITCH_LOW_LOWER = HALF_PERIOD;

      //---------------------------
      // Config filter and loop-mux
      //---------------------------
      LoopMuxRegs.DPWMMUX.bit.DPWM0_FILTER_SEL = 0;
      LoopMuxRegs.DPWMMUX.bit.DPWM1_FILTER_SEL = 0;
      LoopMuxRegs.DPWMMUX.bit.DPWM2_FILTER_SEL = 0;
      LoopMuxRegs.DPWMMUX.bit.DPWM3_FILTER_SEL = 0;

      LoopMuxRegs.FILTERKCOMPA.bit.KCOMP0 = MAXPER;
     // Connect to dpwm2
     LoopMuxRegs.DPWMMUX.bit.DPWM0_SYNC_SEL = 2;
      // Connect to dpwm2
      LoopMuxRegs.DPWMMUX.bit.DPWM1_SYNC_SEL = 2;
      // Connect to dpwm2
      LoopMuxRegs.DPWMMUX.bit.DPWM3_SYNC_SEL = 2;

      Dpwm0Regs.DPWMRESDUTY.bit.RESONANT_DUTY = ((MAXPER + 1) >> 1);
      Dpwm1Regs.DPWMRESDUTY.bit.RESONANT_DUTY = ((MAXPER + 1) >> 1);
      Dpwm2Regs.DPWMRESDUTY.bit.RESONANT_DUTY = ((MAXPER + 1) >> 1);
      Dpwm3Regs.DPWMRESDUTY.bit.RESONANT_DUTY = ((MAXPER + 1) >> 1);

      Dpwm0Regs.DPWMCTRL2.bit.FILTER_DUTY_SEL = 2;
      Dpwm1Regs.DPWMCTRL2.bit.FILTER_DUTY_SEL = 2;
      Dpwm2Regs.DPWMCTRL2.bit.FILTER_DUTY_SEL = 2;
      Dpwm3Regs.DPWMCTRL2.bit.FILTER_DUTY_SEL = 2;

      //use dpwm3 period
      LoopMuxRegs.FILTERMUX.bit.FILTER0_PER_SEL = 3;
      //use dpwm3 period
      LoopMuxRegs.FILTERMUX.bit.FILTER1_PER_SEL = 3;

      Dpwm0Regs.DPWMCTRL2.bit.RESON_DEADTIME_COMP_EN = 1;
      Dpwm1Regs.DPWMCTRL2.bit.RESON_DEADTIME_COMP_EN = 1;
      Dpwm2Regs.DPWMCTRL2.bit.RESON_DEADTIME_COMP_EN = 1;
      Dpwm3Regs.DPWMCTRL2.bit.RESON_DEADTIME_COMP_EN = 1;

      //configure phase shift between DPWM modules
       Dpwm0Regs.DPWMPHASETRIG.all =  MINPER * 8;
       Dpwm2Regs.DPWMPHASETRIG.all =  MINPER * 8;

       LoopMuxRegs.FECTRL0MUX.bit.DPWM0_FRAME_SYNC_EN = 1;
       LoopMuxRegs.FECTRL1MUX.bit.DPWM1_FRAME_SYNC_EN = 1;

      Dpwm0Regs.DPWMCTRL0.bit.MIN_DUTY_MODE = 2;
      Dpwm1Regs.DPWMCTRL0.bit.MIN_DUTY_MODE = 2;
      Dpwm2Regs.DPWMCTRL0.bit.MIN_DUTY_MODE = 2;
      Dpwm3Regs.DPWMCTRL0.bit.MIN_DUTY_MODE = 2;

      Dpwm0Regs.DPWMMINDUTYLO.bit.MIN_DUTY_LOW = (DEADTIME3 + 8) >> 4;
      Dpwm0Regs.DPWMMINDUTYHI.bit.MIN_DUTY_HIGH = (DEADTIME3 + 8) >> 4;

      Dpwm1Regs.DPWMMINDUTYLO.bit.MIN_DUTY_LOW = 263;
      Dpwm1Regs.DPWMMINDUTYHI.bit.MIN_DUTY_HIGH = 263;
```

```
        Dpwm2Regs.DPWMMINDUTYLO.bit.MIN_DUTY_LOW = (DEADTIME1 + 8) >> 4;
        Dpwm2Regs.DPWMMINDUTYHI.bit.MIN_DUTY_HIGH = (DEADTIME1 + 8) >> 4;

        Dpwm3Regs.DPWMMINDUTYLO.bit.MIN_DUTY_LOW = (DEADTIME1 + 8) >> 4;
        Dpwm3Regs.DPWMMINDUTYHI.bit.MIN_DUTY_HIGH = (DEADTIME1 + 8) >> 4;

        Dpwm0Regs.DPWMCTRL1.bit.GLOBAL_PERIOD_EN = 1;
        Dpwm1Regs.DPWMCTRL1.bit.GLOBAL_PERIOD_EN = 1;
        Dpwm2Regs.DPWMCTRL1.bit.GLOBAL_PERIOD_EN = 1;
        Dpwm3Regs.DPWMCTRL1.bit.GLOBAL_PERIOD_EN = 1;

        LoopMuxRegs.PWMGLBPER.bit.PRD = MINPER;

        Filter0Regs.FILTERYNCLPLO.all = 0xFFFF;
        Filter1Regs.FILTERYNCLPLO.all = 0xFFFF;

        Filter0Regs.FILTERCTRL.bit.PERIOD_MULT_SEL = 1;
        Filter0Regs.FILTERCTRL.bit.OUTPUT_MULT_SEL = 3;
}
```