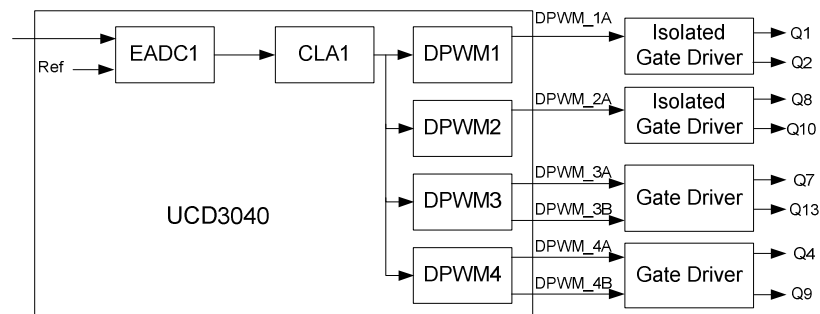
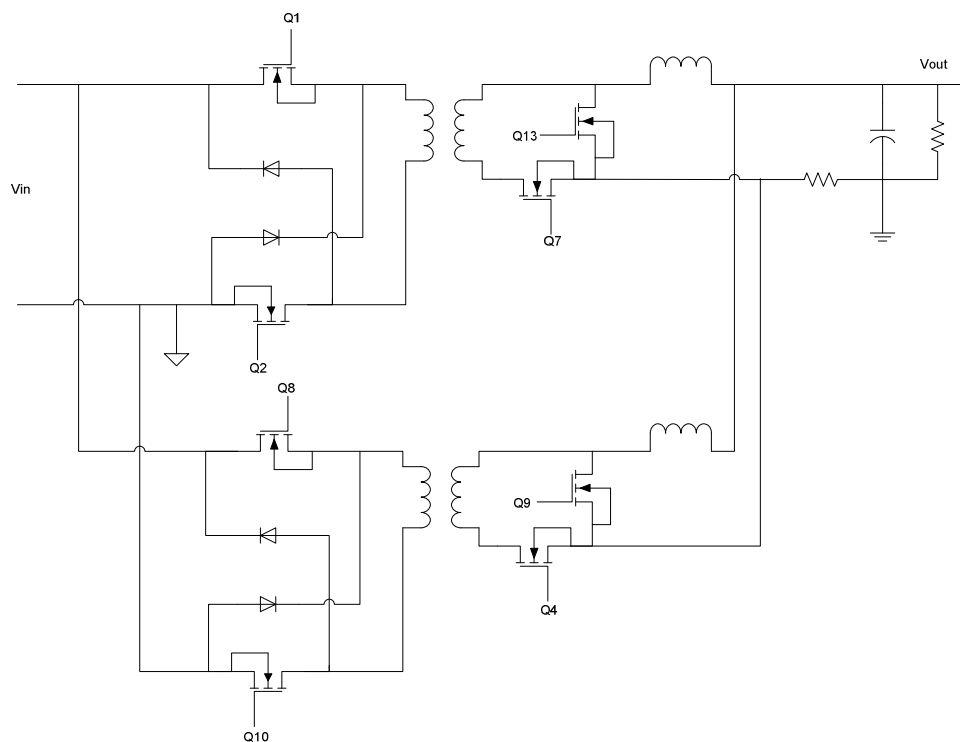
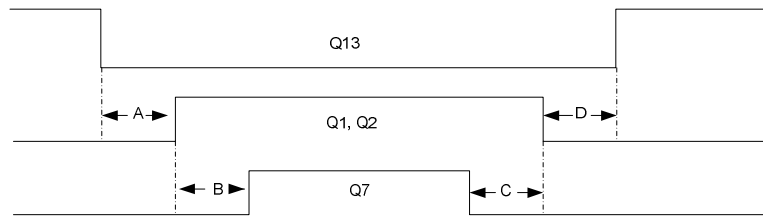


8 DPWM Configuration

8.1 DPWM waveform

This forward converter has two phases running in parallel. Each phase needs 3 PWM signals: the primary MOSFET, the forward MOSFET, and the freewheel MOSFET (sync FET). The second phase has the same PWM waveform as the first one, except that it is 180° phase shifted. The PWM signal connection, waveform and dead time are shown below:





//delays are all in nanoseconds

#define DEAD_TIME_A 20 //delay from sync falling edge to primary rising edge

#define DEAD_TIME_B 100 //delay from primary rising edge to forward rising edge

#define DEAD_TIME_C 40 //delay from forward falling edge to primary falling edge

#define DEAD_TIME_D 270 //delay from primary falling edge to sync rising edge

Be very careful when adjusting the dead time, improper dead time may cause short through and damage the power converter.

8.2 Firmware Setup for DPWM

The firmware setup for the PWM waveform is shown below. The line with “//” ahead is the comment for the code.

//PWM clock(low resolution) is 250Mhz, so 1 tick is 4 nanoseconds

#define LOW_RESOLUTION_DIVIDER 4.0

//the high resolution clock has resolution of 250pSec, so 1 nanosecond equals 4 ticks

#define HIGH_RESOLUTION_MULTIPLIER (16/LOW_RESOLUTION_DIVIDER)

void init_dpwm1(void)

{

//no CLA, leave PWM off for the moment

Dpwm1Regs.DPWMCTRL1.all = 0;

//set PWM period

Dpwm1Regs.DPWMPRD.all = PWM_PERIOD;

//set dead time A as Event 1. Event 1 is low resolution

//Event 2 is set by CLA output

//since we don't use DPWM 1B, so we don't need to set Event 3 and Event 4

Dpwm1Regs.DPWMEV1.all = DEAD_TIME_A/LOW_RESOLUTION_DIVIDER;

Dpwm1Regs.DPWMCTRL1.bit.CLA_CH_SEL = 0; //selected CLA1

//set initial EADC sample trigger point, it will be updated in real time based on PWM duty

//cycle. In general, to avoid noise, it is wise to have the sample trigger as far from switching //edges as possible.

Dpwm1Regs.DPWMSAMPTRIG.all = HIGH_SAMPLE_TRIGGER;

//trigger the second phase with 180 degrees phase shift.

Dpwm1Regs.DPWMPHASETRIG.all = PWM_PERIOD/2;

Dpwm1Regs.EADCCTRL.bit.EADC_ENA = 1; //enable eADC

```

Dpwm1Regs.EADCCTRL.bit.SCFE_ENA = 1; //and switched cap front end.

Dpwm1Regs.EADCDAC.bit.DAC_VALUE = 0x0; //set initial setpoint to 0

Dpwm1Regs.DPWMCTRL1.bit.CLA_ENABLE = 1; //use cla now

//CLA update only at end of period
//The comparisons for events are done on a basis of simple equality, rather than greater
//than or less than, it is necessary to keep the end of CLA calculations away from
//switching times, as a change of the event value at the wrong time could lead to an
//event being missed, and a DPWM pin not being turned on or off when it should
Dpwm1Regs.DPWMCTRL1.bit.UPDATE_END_PRD_ENA = 1;

//for safety purpose, all PWM should be disabled and driven low before the converter
//starts up.
Dpwm1Regs.DPWMCTRL1.bit.GPIO_A_ENA = 1; //disable PWM A, pin driven low
Dpwm1Regs.DPWMCTRL1.bit.GPIO_B_ENA = 1; //disable PWM B, pin driven low

Dpwm1Regs.DPWMCTRL1.bit.PWM_ENA = 1; //and now enable the PWM.
}

void init_dpwm2(void)
{
    Dpwm2Regs.DPWMCTRL1.all = 0; //no CLA, leave PWM off for the moment

    Dpwm2Regs.DPWMPRD.all = PWM_PERIOD; //set PWM period

    //set dead time A. Event 1 is low resolution
    Dpwm2Regs.DPWMEV1.all = DEAD_TIME_A/LOW_RESOLUTION_DIVIDER;

    //this is used to trigger DPWM3 (which is used to control the first phase) back to 0
    //degrees. plus adjustment for primary switch transformer
    Dpwm2Regs.DPWMPHASETRIG.all = PWM_PERIOD/2 + TRANSFORMER_DELAY_ADJUST/LOW_RESOLUTION_DIVIDER;

    //the two phase PWM signals should be synchronized
    Dpwm2Regs.DPWMCTRL1.bit.MSYNC_CH_SEL = 0; //sync to DPWM1
    Dpwm2Regs.DPWMCTRL1.bit.MSYNC_SLAVE_ENA = 1; //enable sync

    //there is only one CLA be used, DPWM2 get input also from CLA1, so we need to
    //disable eADC2.
    Dpwm2Regs.EADCCTRL.bit.EADC_ENA = 0; //disable eADC
    Dpwm2Regs.EADCCTRL.bit.SCFE_ENA = 0; //and switched cap front end.

    Dpwm2Regs.DPWMCTRL1.bit.CLA_CH_SEL = 0; //select CLA1
    Dpwm2Regs.DPWMCTRL1.bit.CLA_ENABLE = 1; //use CLA now

    //CLA update only at end of period
    Dpwm2Regs.DPWMCTRL1.bit.UPDATE_END_PRD_ENA = 1;

    Dpwm2Regs.DPWMCTRL1.bit.GPIO_A_ENA = 1; //disable PWM A, pin driven low
    Dpwm2Regs.DPWMCTRL1.bit.GPIO_B_ENA = 1; //disable PWM B, pin driven low

    Dpwm2Regs.DPWMCTRL1.bit.PWM_ENA = 1; //now enable the PWM.
}

```

```

void init_dpwm3(void)
{
    Dpwm3Regs.DPWMCTRL1.all = 0; //no CLA, leave PWM off for the moment

    Dpwm3Regs.DPWMPRD.all = PWM_PERIOD; //set PWM period

    //Event 1 starts at beginning of period, with room for dead time = A + B
    Dpwm3Regs.DPWMEV1.all = (DEAD_TIME_A + DEAD_TIME_B)/LOW_RESOLUTION_DIVIDER;

    //set Event 2 at half the period (P/2)*16, Event 2 is high resolution
    //this is the initial value for Event 2, it combines with Event 3 to set the delay between the
    //falling edge on DPWMA and the rising edge on DPWMB. The real Event 2 is set by CLA
    //output
    Dpwm3Regs.DPWMEV2.all = PWM_PERIOD*8;

    //the delay between DPWM 3A and DPWM 3B should be (D + C)
    //but DPWM 3A has a cycle adjustment of -(B + C)
    //to maintain the required delay, Event 3 also need to be adjusted by -(B + C), which
    //gives D + C - (B + C) = D - B
    Dpwm3Regs.DPWMEV3.all = (PWM_PERIOD*8) + ((DEAD_TIME_D - DEAD_TIME_B)*HIGH_RESOLUTION_MULTIPLIER);

    //B falling just before A rises again
    Dpwm3Regs.DPWMEV4.all = (PWM_PERIOD*16) - 1 ;

    //this is used to trigger DPWM4 (which is for the second phase) with a 180 degree phase
    //shift
    Dpwm3Regs.DPWMPHASETRIG.all = PWM_PERIOD/2;

    //DPWM 3A pulse width equals to (CLA - B - C), this is done through cycle //adjustment
    Dpwm3Regs.DPWMCYCADJA.all = -((DEAD_TIME_B + DEAD_TIME_C)*HIGH_RESOLUTION_MULTIPLIER);

    Dpwm3Regs.DPWMCLFCTRL.all = 0; //disable current limit flag logic
    Dpwm3Regs.DPWMCTRL1.bit.CLA_ENABLE = 1; //use CLA now

    Dpwm3Regs.EADCCTRL.bit.EADC_ENA = 0; //disable eADC
    Dpwm3Regs.EADCCTRL.bit.SCFE_ENA = 0; //and switched cap front end.

    Dpwm3Regs.DPWMCTRL1.bit.MSYNC_SLAVE_ENA = 1; //enable sync
    Dpwm3Regs.DPWMCTRL1.bit.CLA_CH_SEL = 0; //select CLA1
    Dpwm3Regs.DPWMCTRL1.bit.MSYNC_CH_SEL = 1; //sync to DPWM2

    //CLA update only at end of period
    Dpwm3Regs.DPWMCTRL1.bit.UPDATE_END_PRD_ENA = 1;

    Dpwm3Regs.DPWMCTRL1.bit.GPIO_A_ENA = 1; //disable PWM A, pin driven low
    Dpwm3Regs.DPWMCTRL1.bit.GPIO_B_ENA = 1; //disable PWM B, pin driven low

    Dpwm3Regs.DPWMINT.bit.PRD_INT_SCALE = 2; //interrupt every 4 cycles

    Dpwm3Regs.DPWMCTRL1.bit.PWM_ENA = 1; //and now enable the PWM.
}

```

```

void init_dpwm4(void)
{
    Dpwm4Regs.DPWMCTRL1.all = 0; //no CLA, leave PWM off for the moment

    Dpwm4Regs.DPWMPRD.all = PWM_PERIOD; //set PWM period

    //start at beginning of period, with room for dead time = A + B
    Dpwm4Regs.DPWMEV1.all = (DEAD_TIME_A + DEAD_TIME_B)/LOW_RESOLUTION_DIVIDER;

    //set Event 2 at half the period (P/2)*16, Event 2 is high resolution
    //this is the initial value for Event 2, it combines with Event 3 to set the delay between the
    //falling edge on DPWMA and the rising edge on DPWMB. The real Event 2 is set by CLA
    //output
    Dpwm4Regs.DPWMEV2.all = PWM_PERIOD*8;

    //the delay between DPWM 4A and DPWM 4B should be (D + C)
    //but DPWM 4A has a cycle adjustment of -(B + C)
    //to maintain the required delay, Event 3 also need to be adjusted by -(B + C), which
    //gives D + C - (B + C) = D - B
    Dpwm4Regs.DPWMEV3.all = (PWM_PERIOD*8) + ((DEAD_TIME_D - DEAD_TIME_B)*HIGH_RESOLUTION_MULTIPLIER);

    //B falling just before A rises again
    Dpwm4Regs.DPWMEV4.all = (PWM_PERIOD*16) - 1 ;

    //DPWM 3A pulse width equals to (CLA - B - C), this is done through cycle //adjustement
    Dpwm4Regs.DPWMCYCADJA.all = -((DEAD_TIME_B + DEAD_TIME_C)*HIGH_RESOLUTION_MULTIPLIER);

    Dpwm4Regs.DPWMCLFCTRL.all = 0; //disable current limit flag logic
    Dpwm4Regs.DPWMCTRL1.bit.CLA_CH_SEL = 0; //select CLA1
    Dpwm4Regs.DPWMCTRL1.bit.CLA_ENABLE = 1; //use cla now

    Dpwm4Regs.EADCCTRL.bit.EADC_ENA = 0; //disable ead
    Dpwm4Regs.EADCCTRL.bit.SCFE_ENA = 0; //and switched cap front end.

    Dpwm4Regs.DPWMCTRL1.bit.MSYNC_SLAVE_ENA = 1; //enable sync
    Dpwm4Regs.DPWMCTRL1.bit.MSYNC_CH_SEL = 2; //sync to DPWM3

    //cla update only at end of period
    Dpwm4Regs.DPWMCTRL1.bit.UPDATE_END_PRD_ENA = 1;

    Dpwm4Regs.DPWMCTRL1.bit.GPIO_A_ENA = 1; //disable PWM A, pin driven low
    Dpwm4Regs.DPWMCTRL1.bit.GPIO_B_ENA = 1; //disable PWM B, pin driven low

    Dpwm4Regs.DPWMCTRL1.bit.PWM_ENA = 1; //and now enable the PWM.
}

```