UCD31xx Application Note Wide Frequency Range DPWM Synchronization on the Fly

Literature Number: xxxxxx

9/24/2012

Table of Contents

1 Introduction 2
2 Reset the DPWM period counter at a proper place 2
2.1 Two principles 2
2.2 Improper reset 3
2.3 Find a safe place to reset 4
3 Frequency and phase measurement using T24 capture 7
3.1 T24 timer capture structure 7
3.2 Frequency measurement 8
3.3 Phase measurement 8
4 Frequency and phase adjustment 10
4.1 Frequency adjustment 10
4.2 Phase adjustment 11
5 HSFB DPWM synchronization example 11
5.1 Firmware flowchart 11
5.2 DPWM configuration 11
5.3 Frequency measurement and adjustment 13
5.4 Turn on external sync 13
5.5 Phase adjustment 14
6 Test waveforms 14

1 Introduction

UCD31xx has a SYNC pin to synchronize DPWM waveforms between multiple UCD31xx devices. The SYNC pin of the master is configured as SYNC out, while the SYNC pin of the slave is configured as SYNC in. Two UCD31xx based power supplies can be connected during normal operation (hot sync) if carefully configured and programmed.

The DPWM module in UCD31xx is based on a 14 bit counter with period and frequency control. When the external synchronization signal is received from SYNC pin, and the DPWM module is enabled to accept external SYNC signal, the DPWM counter will reset to zero or a preset value. In this way, the DPWM signal is synchronized to an external SYNC signal.

Improper reset of the DPWM counter can result in pulse extension or shoot through. This application note suggests a right procedure of “hot sync”. The basic idea is to measure the master’s frequency and then adjust the frequency of the slave. After the frequency is properly adjusted, turn on external synchronization bit of the slave. This application note first introduces the properties of UCD31xx’s SYNC pin, and how to do frequency and phase measurement using the 24-bit timer capture. Then explained in detail about how DPWM synchronization between two hard switching full bridge converters is achieved. The last section shows the test waveforms.

The proposed method supports wide frequency range DPWM synchronization on UCD31xx based hard switching full bridge converter. The synchronization can be down during normal operation with little Vout drop only at the transition point.

2 Reset the DPWM period counter at a proper place

2.1 Two principles

The period counter of the DPWM module is reset to zero or a preset value (determined by DPWMCNTPRE) by three events:

1. DPWM enable
2. SYNC received (slave mode enabled)
3. Counter reaches period register value

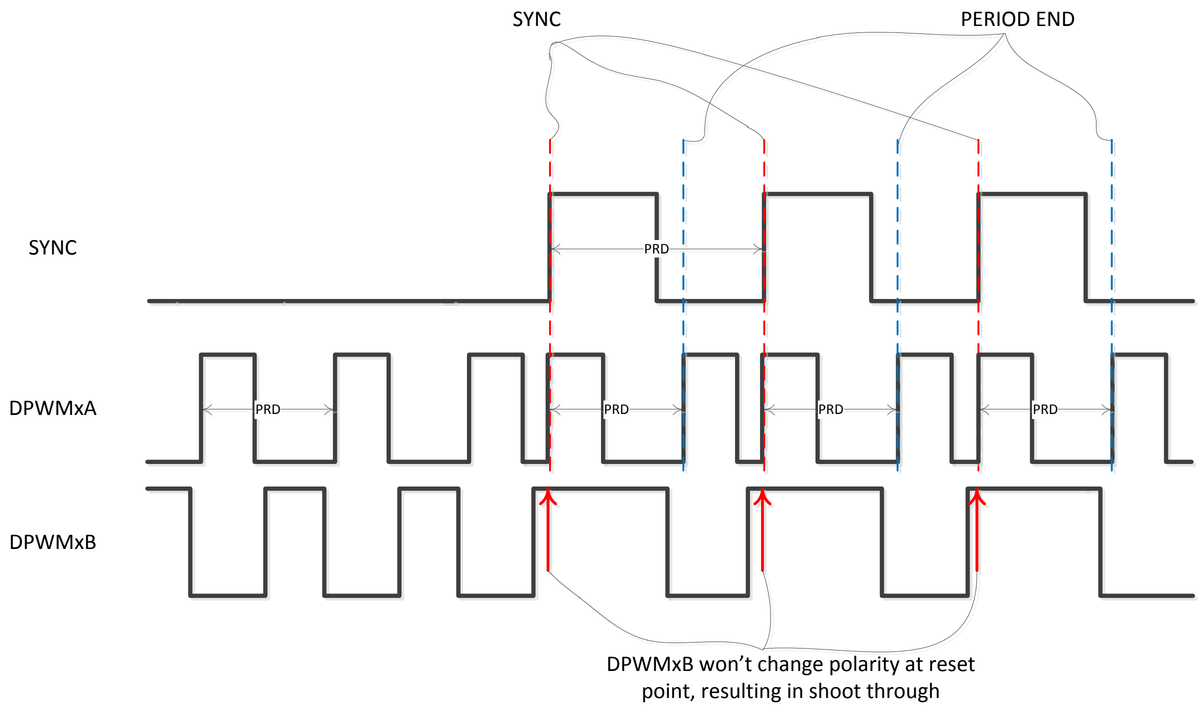
Principle 1:
The first rising edge (EV1 typically) of a pair of DPWM signal will come out right after one of the above three events happens, and it depends on which event comes in first.

Principle 2:
The DPWM pulses won't change their polarities at the reset point until the next DPWM EVENT or reset event comes in. For example, if at the reset point, DPWMxA is high and DPWMxB is low, after resetting, DPWMxA will keep high and DPWMxB will keep low, until the next DPWM EVENT happens.

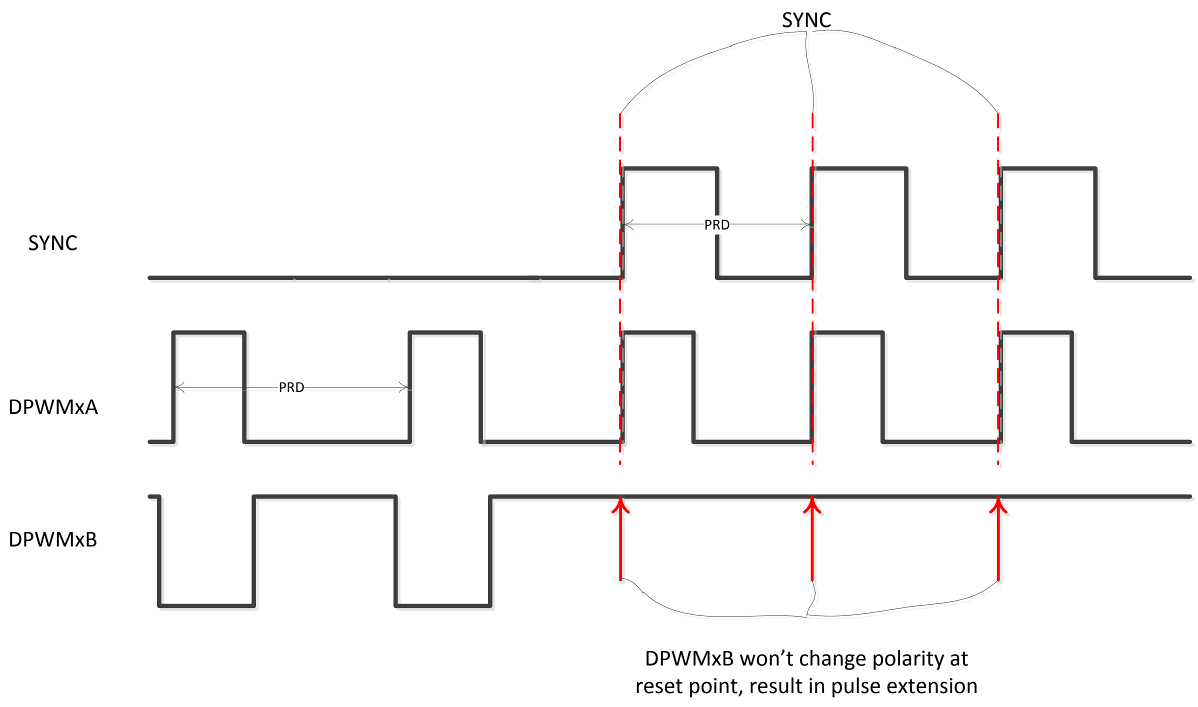
2.2 Improper reset

Improper reset may result in pulse extension or multiple pulses in one cycle.
Let's see some examples based on the two effects:

Example 1:
In this example, the SYNC frequency is lower than the DPWM frequency. And the pulse width of DPWMxA is relatively small. The first SYNC signal comes in between EV3 and EV4. At this moment, DPWMxA is low and DPWMxB is high. Since EV1 is put at the beginning of the period, right after the SYNC signal comes in, DPWMxA becomes high. But DPWMxB will remain high. This results in shoot through. Since the frequency of the SYNC signal is much lower than the DPWM frequency, the end of period reset will happen before the next SYNC signal comes in. So there will be two pulses on DPWMxA in one cycle.


SYNC
PERIOD END
SYNC
PRD
DPWMxA
PRD
PRD
PRD
PRD
DPWMxB
DPWMxB won't change polarity at reset point, resulting in shoot through

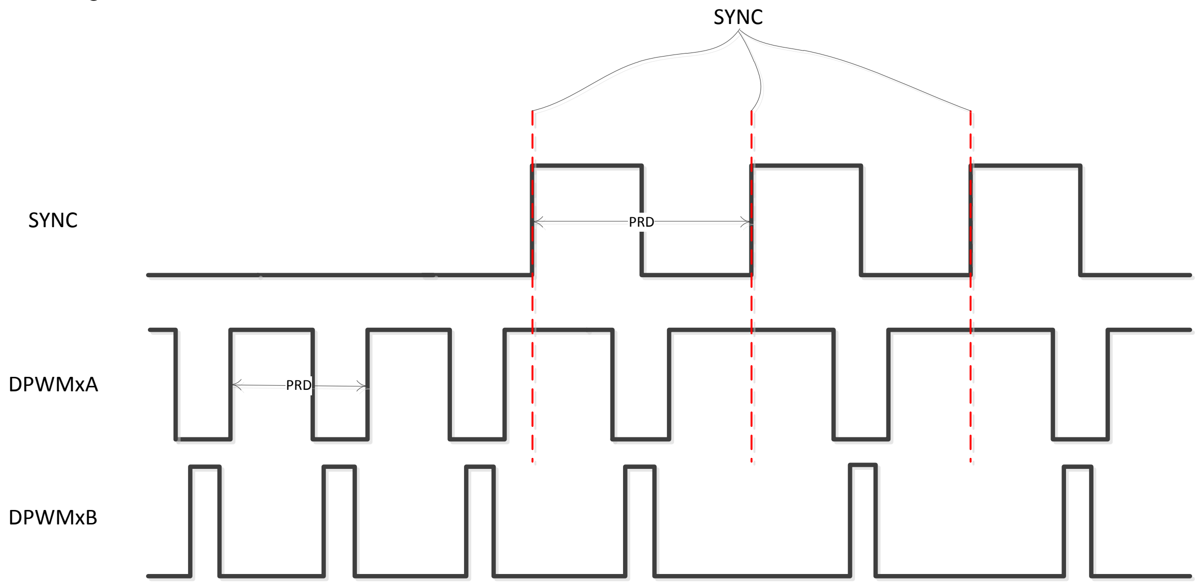
Example 2:
In this example, the frequency of SYNC signal is higher than the DPWM frequency. The first SYNC signal comes in between EV3 and EV4, when DPWMxB is high. Because the frequency of SYNC is higher than the DPWM frequency, the following SYNC signal will always be between EV3 and EV4, and DPWMxB will never shut off.


SYNC
SYNC
PRD
DPWMxA
PRD
DPWMxB
DPWMxB won't change polarity at reset point, result in pulse extension

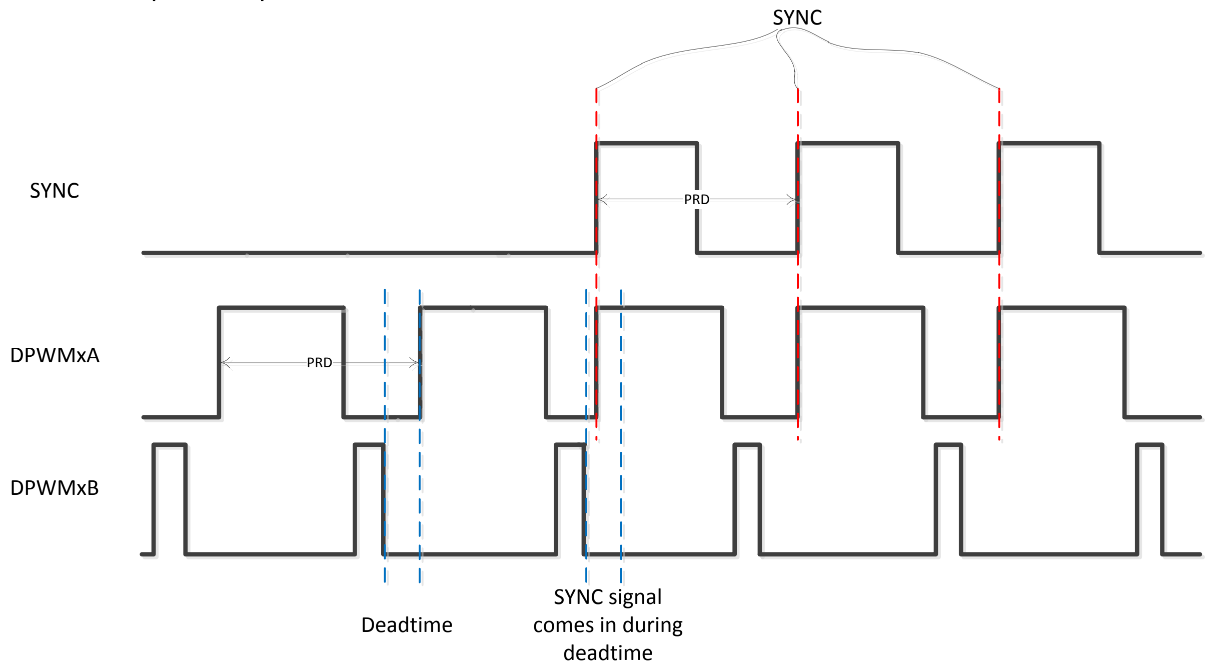
2.3 *Find a safe place to reset*

The following examples show two configurations that won't cause pulse extension or shoot through.

Example 3:
The frequency of SYNC is slightly lower than the frequency of DPWMs. SYNC signal comes in between EV1 and EV2. After synchronization, DPWMxA pulses become wider. There is no pulse extension or shoot through.



Example 4:
The frequency of SYNC is almost the same as the DPWM frequency. SYNC comes in between EV4 and EV1. There is no pulse extension or shoot through. The first pulse after synchronization becomes a little bit closer to the previous pulse.



Among the above examples, the configuration in Example 4 gives the least distortion. But it requires very precise frequency and phase measurement in order to put the SYNC signal at the best place. The timer capture module on UCD31xx has a resolution of 64ns. Compared with deadtime (EV4~EV1 window), it is not fine enough. This configuration is not practical in most applications, because the “safe area” is too narrow.

Example 3 has no pulse extension or shoot-through. But it makes DPWMxA wider after synchronization. This is what we don’t want to see in a power supply.

The proposed method uses a spare DPWM to synchronize with the external signal. The other DPWMs are slaves of the spare DPWM.

There are several benefits of using a spare DPWM:

(1) The spare DPWM is not used for power stage drivers. So the configuration of it has fewer constraints and we can make a wide “safe area” for the SYNC signal.
(2) The power stage DPWMs are slaves to the spare DPWM all the time. They are the same frequency and have a fixed phase relationship. If there is no pulse extension or shoot-through on the spare DPWM, the power stage DPWMs will be safe. And because of the fixed frequency and phase relationship, the power stage DPWMs don’t need to deal with a “moving master” which could cause pulse distortion problem.

The following figure shows the configuration of the spare DPWM. To avoid pulse extension/shoot-through, the “safe area” is made very wide. And the unsafe area is blanked in the firmware. Here are the key points in configuration:

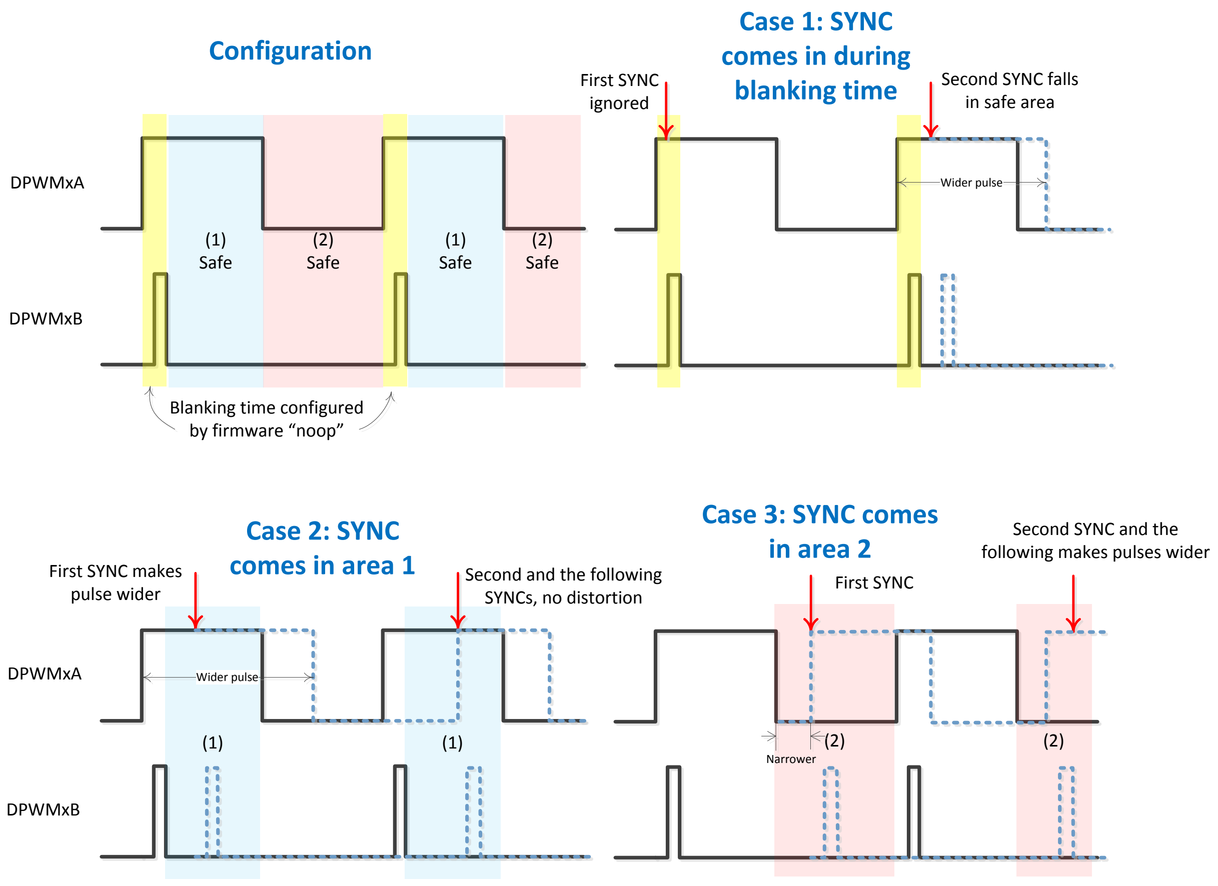
(1) EV1, EV3, and EV4 are very close to the beginning of the period, as shown in the figure. For example, EV1 = 20, EV3 = 100, EV4 = 200. The duty of DPWMxA is 50%. EV1~EV4 window is blanking time.

(2) The frequency of SYNC signal is slightly lower than the frequency of the DPWMs. For example, if the DPWM period is PRD_DPWM, the SYNC period is PRD_SYNC. Then they should meet the following equation:

PRD_DPWM < PRD_SYNC + blanking time

In this way, if the first SYNC signal comes in during blanking time, the second SYNC signal will be pushed to the "safe area".

(3) The EV1~EV4 window is blanked in firmware – EXT_SYNC_EN bit won't be turned on in this area. If the SYNC signal comes in the blanking area, it is ignored. The blanking time is set by firmware delay. The delay can be set this way: at the beginning of a period – rising edge of DPWMxA, generate a fast interrupt (DPWM period interrupt), put some delay which is larger than EV4 – EV1 time window in the interrupt, and then turn on EXT_SYNC_EN. In fact, if EV1~EV4 window is made very small, it is automatically blanked by the interrupt entrance delay.


Configuration
DPWMxA
DPWMxB
(1) Safe
(2) Safe
(1) Safe
(2) Safe
Blanking time configured by firmware "noop"
Case 1: SYNC comes in during blanking time
First SYNC ignored
Second SYNC falls in safe area
Wider pulse
Case 2: SYNC comes in area 1
First SYNC makes pulse wider
Second and the following SYNCs, no distortion
Wider pulse
(1)
(1)
DPWMxA
DPWMxB
Case 3: SYNC comes in area 2
First SYNC
Second SYNC and the following makes pulses wider
Narrower
(2)
(2)

Let's see three different cases that may happen to this DPWM configuration:
(the blue line in the figure is the waveform after sync signal is received)

Case 1: SYNC comes in during blanking time
The first SYNC is ignored, and the second SYNC will fall in area 1, due to the PRD_DPWM < PRD_SYNC + blanking time relationship.

Case 2: SYNC comes in area 1
The first DPWMxA pulse gets wider; the following waveforms have no distortion.

Case 3: SYNC comes in area 2
The first SYNC makes the DPWMxA pulse closer to the previous pulse; the following DPWMxA pulses are a little bit wider because of the period difference. But this wider pulse won't cause distortion on the

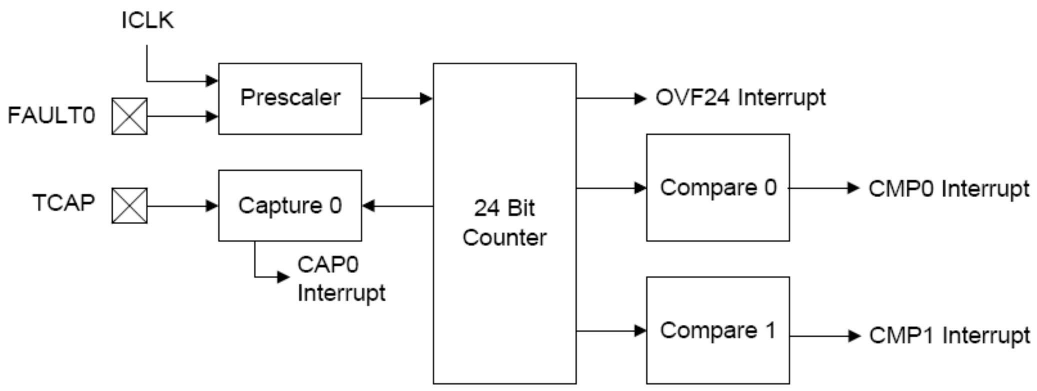
power stage DPWMs, because what matters is just the starting point of this spare DPWM, but not the pulse width.

As we can see, only at the transition point, the spare DPWM gets a little bit. After that, the spare DPWM will provide stable reference to the other DPWMs.

3 Frequency and phase measurement using T24 capture

3.1 *T24 timer capture structure*

The core of the T24 timer is a 24-bit free-running counter. The clock input for the T24 timer can come from a clock within the UCD31xx, ICLK, which is normally 15.6MHz. It can also be used with an external clock from the FAULT0 pin. In this application, the internal ICLK is used, which gives a 64ns time resolution.



The 24-bit timer has a capture block. It can be programmed to capture the 24-bit timer value on an edge on the external signal. The edge control bits can select from rising edge, falling edge, or both edges. To measure frequency/pulse width/phase, it is required to read the capture data at a proper time. So the T24 capture interrupt is enabled by setting the bit CAP_INT_ENA, and it is configured to be fast interrupt. The capture module input can be selected from TCAP pin, SCI_RX[0], SCI_RX[1], and SYNC pin. In this application, the timer capture is connected to SYNC pin. The SYNC pin can be configured as SYNC in or SYNC out. When configured as SYNC in, the timer capture can capture the edges from the other device. When configured as SYNC out (SYNC out source can be selected to be any DPWM output), the timer capture catches its own edges.
Here is the timer capture initialization:

```
void init_timer_capture(void)
{
	TimerRegs.T24CAPCTRL.bit.CAP_INT_ENA = 1; //enable capture interrupt
	TimerRegs.T24CAPCTRL.bit.EDGE = 1; //capture on posedge
	TimerRegs.T24CAPCTRL.bit.CAP_SEL = 3; //connect T24 capture to SYNC pin
}
```

To enable timer capture interrupt, do the following in the interrupt initialization:

```
	write_reqmask(CIMINT_ALL_TMR_CAPT0);
	write_firqpr(CIMINT_ALL_TMR_CAPT0);

	enable_fast_interrupt();
	enable_interrupt();
```

3.2 Frequency measurement

For frequency measurement, two sequential positive edges are captured, and the time difference is calculated. The following needs to be included in the fast interrupt to perform frequency measurement:

```
//read capture data
t24_latched = TimerRegs.T24CAPDAT.bit.CAP_DAT;

//calculate the counter difference between the current and
//the previous capture data
//the "else" is to handle overflow
if(t24_latched > t24_latched_previous)
{
      t24_latched_diff = t24_latched - t24_latched_previous;
}
else
{
      t24_latched_diff = t24_latched + 0xFFFFFF - t24_latched_previous;
}

//save capture data to t24_latched_previous
t24_latched_previous = t24_latched;
//moving average to calculate the time difference in steps of 250ps
avg_prd_in_250ps =t24_latched_diff + avg_prd_in_250ps-(avg_prd_in_250ps>>8);

t24_int_counter ++;

//clear the interrupt flag
TimerRegs.T24CAPCTRL.bit.CAP_INT_FLAG = 0;
```

3.3 Phase measurement

To measure the phase difference between two DPWMs or master and slave, the timer capture module first capture the rising edge of the first DPWM (or slave), then capture the rising edge of the second DPWM (or master).

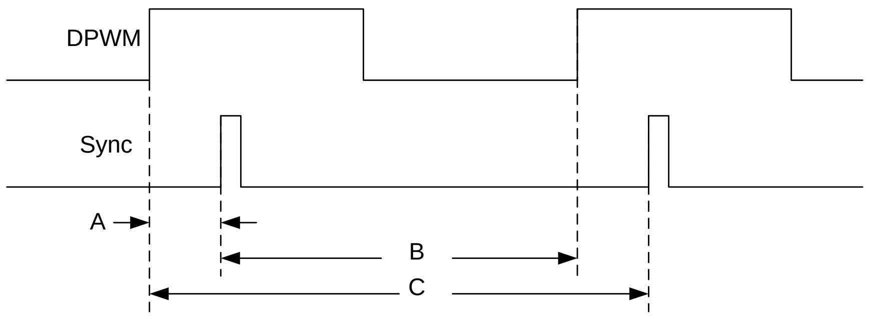
UCD31xx has one SYNC module, but the module can be routed to 3 different pins (2 different pins on 40pin package) by selecting the IOMUX bits. In phase measurement application, the TCK pin is connected to DPWM module of the salve, SYNC pin is connected to the master SYNC out; the SYNC module switches between TCK pin and SYNC pin.

The following is the code that should be included in the interrupt. ms_switch is a switch flag between master and slave. ms_switch == 0 means slave and ms_switch == 1 means master. Note that only after the capture data on the salve is read, we calculate the difference between the two capture.

```
//read capture data
t24_latched = TimerRegs.T24CAPDAT.bit.CAP_DAT;
//clear the interrupt flag
TimerRegs.T24CAPCTRL.bit.CAP_INT_FLAG = 0;
//master slave switch is 0, switch to master
if(ms_switch == 0)
{
      //sync pin configured as input
      LoopMuxRegs.SYNCCTRL.bit.SYNC_DIR = 1;
      //sync pin function as sync
      //TCK pin configured as TCK
```

```
        MiscAnalogRegs.IOMUX.bit.SYNC_MUX_SEL = 0;
        MiscAnalogRegs.IOMUX.bit.JTAG_CLK_MUX_SEL = 0;

        //switch to master
        ms_switch = 1;

        //difference calculation
        if(t24_latched > t24_latched_previous)
        {
              t24_latched_diff = t24_latched - t24_latched_previous;
        }
        else
        {
              t24_latched_diff = t24_latched + 0xFFFFFF - t24_latched_previous;
        }

        //averaging
        avg_ph_in_250ps=t24_latched_diff+avg_ph_in_250ps-(avg_ph_in_250ps>>8);
}

//if master slave switch is 1, switch to slave
else if(ms_switch == 1)
{
        //sync pin configured as another function
        //TCK pin configured as sync
        MiscAnalogRegs.IOMUX.bit.SYNC_MUX_SEL = 2;
        MiscAnalogRegs.IOMUX.bit.JTAG_CLK_MUX_SEL = 2;
        //sync pin configured as output, a DPWM module output is connected to
        //sync pin at initialization
        LoopMuxRegs.SYNCCTRL.bit.SYNC_DIR = 0;
        ms_switch = 0;
}
```

Entering and exiting the interrupt takes time. If the phase difference is very small, say less than 30*250ps. There will be missing pulses. For example, in the following figure, we are measuring (A), but (A) is too small and what we get is (C). (C) is (A) plus one period. So in the firmware, we manually subtract the period value from (C) and get (A).



Here is the code:

```
//if phase shift is very small, one pulse is missed,
//subtract result with period. to get the correct phase value
if(avg_ph_in_250ps > avg_prd_in_250ps)
{
        avg_ph_in_250ps = avg_ph_in_250ps - avg_prd_in_250ps;
}
```

4 Frequency and phase adjustment

4.1 Frequency adjustment

The frequency adjustment function frequency_switch( ) is called every 100us in the standard interrupt. And change the frequency and other frequency related registers step by step (increase or decrease by one at a time). Two global variables period_new and period_old are defined to record the frequency change. period_new can be changed either by measurement of master's frequency or changing the switching frequency on the GUI. For frequency changing on the GUI, please refer to another document: "UCD31xx Test Report – Change Switching Frequency on the Fly".

The following is the frequency_switch( ) function:

```
inline void frequency_switch(void)
{
        if(period_new > period_old)
        {
                pmbus_dcdc_config[0].period = period_old + 1;
                period_old ++;
                configure_vout_ramp_rate();
                configure_ton_rise();
                init_dpwm0();
                init_dpwm1();
                init_dpwm3();
                configure_dpwm_timing();
                configure_others();
        }

        if(period_new < period_old)
        {
                pmbus_dcdc_config[0].period = period_old - 1;
                period_old --;
                configure_vout_ramp_rate();
                configure_ton_rise();
                init_dpwm0();
                init_dpwm1();
                init_dpwm3();
                configure_dpwm_timing();
                configure_others();

        }
}
```

Note that in function configure_others(), the following registers are reconfigured:

1. DPWM global period
2. KCOMP value
3. Filter output clamp

Here is the code:

```
LoopMuxRegs.PWMGLBPER.all = pmbus_dcdc_config[0].period;
LoopMuxRegs.FILTERKCOMPB.bit.KCOMP2 = pmbus_dcdc_config[0].period >> 4;
LoopMuxRegs.FILTERKCOMPA.bit.KCOMP0 = (3 * pmbus_dcdc_config[0].period) >> 6;
```

```
Filter0Regs.FILTEROCLPHI.bit.OUTPUT_CLAMP_HIGH = (pmbus_dcdc_config[0].period
>> 1) - 500; // clamp to 50% duty - 500 * 250ps
Filter1Regs.FILTEROCLPHI.bit.OUTPUT_CLAMP_HIGH = (pmbus_dcdc_config[0].period
>> 1) - 500; // clamp to 50% duty - 500 * 250ps
Filter2Regs.FILTEROCLPHI.bit.OUTPUT_CLAMP_HIGH = pmbus_dcdc_config[0].period;
Filter2Regs.FILTEROCLPLO.bit.OUTPUT_CLAMP_LOW = pmbus_dcdc_config[0].period *
0.4;
```

4.2 *Phase adjustment*

Phase adjustment is accomplished by adjusting the DPWM counter preset register DPWMCNTPRE. If enabled, the DPWMCNTPRE register is loaded into the period counter at DPWM startup or on a rising sync edge.

Here is the example code to shift the waveform by 180°.

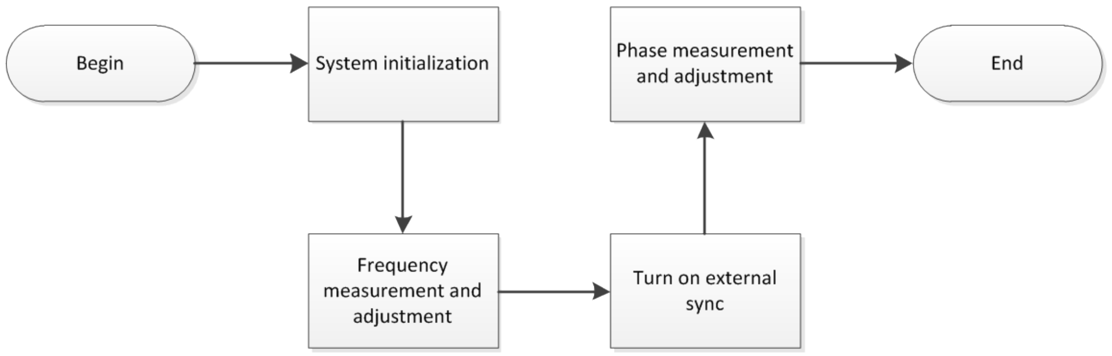
```
Dpwm1Regs.DPWMCNTPRE.all = pmbus_dcdc_config[0].period >> 1;
Dpwm1Regs.DPWMCTRL1.bit.PRESET_EN = 1;    //Enable phase shift
```

Both master and slave mode DPWM modules can use this register. The above example code can be used for initialization. But for safe operation of the power stage, if phase needs to be adjusted on the fly, it is recommended to adjust step by step as well.

5 HSFB DPWM synchronization example

5.1 *Firmware flowchart*

The firmware flowchart of HSFB DPWM synchronization is shown below. The flowchart is for slave only. The master's firmware is very easy: the only thing is to configure SYNC pin as output and select one DPWM for sync out. The slave's firmware contains 4 stages: after system initialization, the slave measures the master's frequency through SYNC pin by T24 capture module. Then adjust the frequency of the slave to a little bit higher than the master's frequency (chapter 2 has explanation). After the frequency adjustment is done, turn off SYNC interrupt and turn on DPWM period interrupt. After blanking time, turn on external synchronization. Finally, measure the phase difference between master and slave, and adjust the phase by adjusting the period counter reset register.


Begin
System initialization
Phase measurement and adjustment
End
Frequency measurement and adjustment
Turn on external sync

5.2 *DPWM configuration*

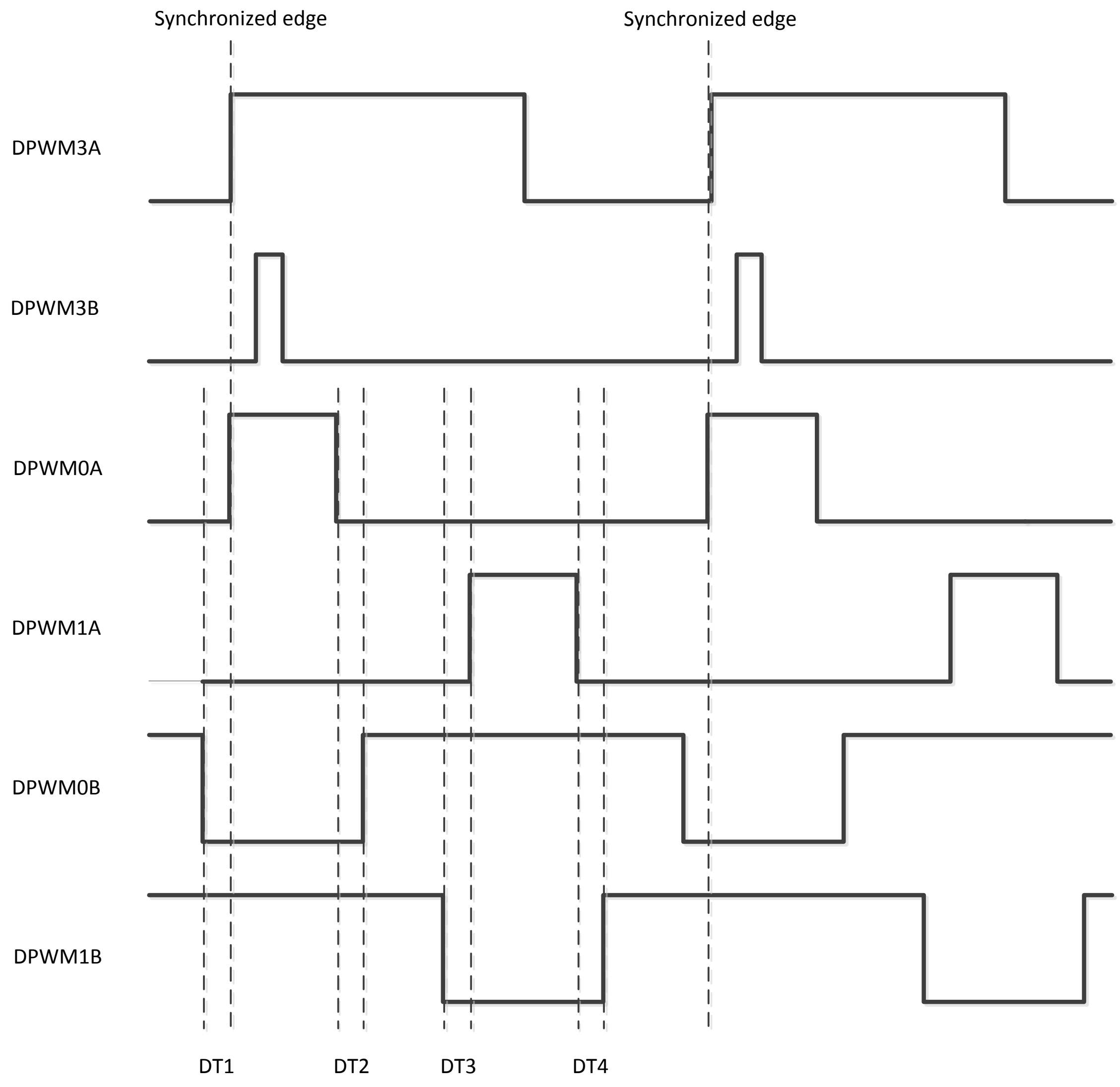
The DPWM synchronization functionality is tested on UCD31xx 360W 12V hard switching full bridge EVM. The detailed hardware connection can be found in the EVM's user guide.

The HSFB configuration uses 2 DPWM pairs:
DPWM0A and DPWM1A - primary side drive;

DPWM0B and DPWM1B – synchronous rectifier (SR) drive.

For synchronization purpose, a third DPWM pair is used – DPWM3. At initialization, DPWM3 is configured as master. DPWM0 and DPWM1 are slaves of DPWM3. After the slave power supply is ready for synchronization, the EXT_SYNC_EN bit in DPWM3 will be turned on, and sync with an external signal.

The configuration of DPWM3 follows what was proposed in Chapter 2. The other DPWM configurations can be found in the EVM user's guide and firmware development guide.



The event settings, protection settings, and sample trigger settings of DPWM0 and DPWM1 are the same as the normal configuration as describe in the user's guide and firmware development manual. Here only specific settings for synchronization purpose are mentioned.

DPWM0 is a slave of DPWM3:

```
    Dpwm0Regs.DPWMCTRL0.bit.MSYNC_SLAVE_EN =1;
    LoopMuxRegs.DPWMMUX.bit.DPWM0_SYNC_SEL =3;
```

DPWM1 is a slave of DPWM3:

```
	Dpwm1Regs.DPWMCTRL0.bit.MSYNC_SLAVE_EN =1;
	LoopMuxRegs.DPWMMUX.bit.DPWM1_SYNC_SEL =3;
```

DPWM3 is initialized as below:

```
void init_dpwm3(void)
{
	Dpwm3Regs.DPWMCTRL0.bit.CLA_EN = 0; //open loop, not controlled by CLA

	Dpwm3Regs.DPWMPRD.all = pmbus_dcdc_config[0].period;
	Dpwm3Regs.DPWMEV1.all = 20;  //at the beginning of period
	Dpwm3Regs.DPWMEV2.all = pmbus_dcdc_config[0].period/2; //50% duty
	Dpwm3Regs.DPWMEV3.all = 200;    //narrow pulse
	Dpwm3Regs.DPWMEV4.all = 400;  //narrow pulse

	Dpwm3Regs.DPWMCTRL1.bit.GLOBAL_PERIOD_EN = 1;
	Dpwm3Regs.DPWMCTRL0.bit.MSYNC_SLAVE_EN = 0; //DPWM3 is master at init
}
```

5.3 *Frequency measurement and adjustment*

After initialization of DPWMs, DPWM3 is master; DPWM0 and DPWM1 are slaves of DPWM3. EXT_SYNC_EN bit of DPWM3 is off.

Then T24 capture is initialized and waits for SYNC signal coming in from SYNC pin. Don't turn on EXT_SYNC_EN at this moment, first measure the frequency of the SYNC signal. The T24 capture frequency measurement method is explained in Chapter 3. The firmware takes 3000 samples of the SYNC frequency and avg_prd_in_250ps is the average period value in steps of 250ps. After 3000 times measurement is done. Change the global variable period_new, so that the function frequency_switch( ) will make changes to the DPWM frequency step by step.

```
if(t24_int_counter == 3000)
{
	period_new = avg_prd_in_250ps - 500;
	t24_flag = 1;
}
```

In this code, the new period of the slave is adjusted to 500x250ps less than the SYNC period, as requirements mentioned in Chapter 2. The variable t24_flag is just a flag to indicate the frequency measurement is done.

After the frequency adjustment is done by frequency_switch( ), another flag will be set:

```
freq_switch = 1;
```

5.4 *Turn on external sync*

After frequency measurement and adjustment is done, turn off timer capture interrupt and leave only the DPWM period interrupt on.

```
TimerRegs.T24CAPCTRL.bit.CAP_INT_ENA = 0;
```

Because EV1~EV4 time window is set to a very small value (400x250ps = 100ns, which is just 3 instruction cycle), the blanking time is automatically achieved by interrupt entrance delay. So just turn on the EXT_SYNC_EN bit in the DPWM3 period fast interrupt, if the requirements are met.

```
if((freq_switch == 1)&&(Dpwm3Regs.DPWMCTRL1.bit.EXT_SYNC_EN == 0)&&(t24_flag
== 1))
{
        MiscAnalogRegs.GLBIOVAL.bit.TDO_IO_VALUE = 1;
        Dpwm3Regs.DPWMCTRL0.bit.MSYNC_SLAVE_EN = 1;
        Dpwm3Regs.DPWMCTRL1.bit.EXT_SYNC_EN = 1;
}
```

The TDO_IO_VALUE bit is to control a GPIO to indicate the turning on of EXT_SYNC_EN, so that the transition point can be captured on a oscilloscope.

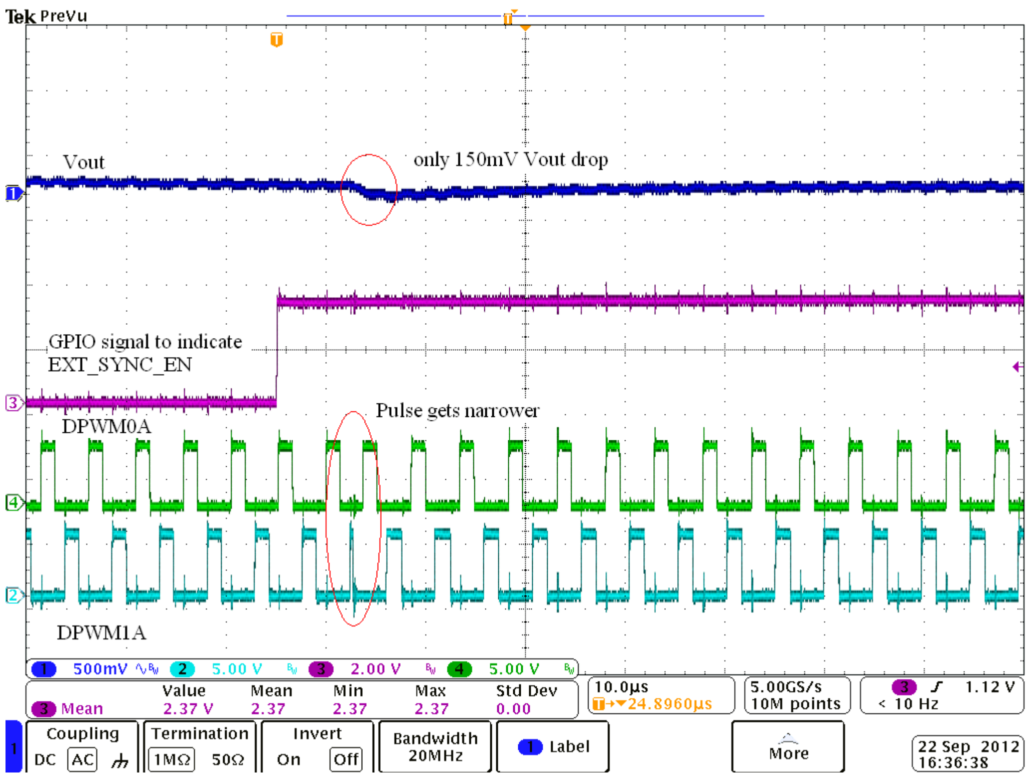
5.5 *Phase adjustment*

The SYNC signal can come in at any place of DPWMs except the blanking time. So after the synchronization is done, phase adjustment is needed to put the master and slave in phase. Phase measurement method is explained in Chapter 3, and the adjustment method is explained in Chapter 4.

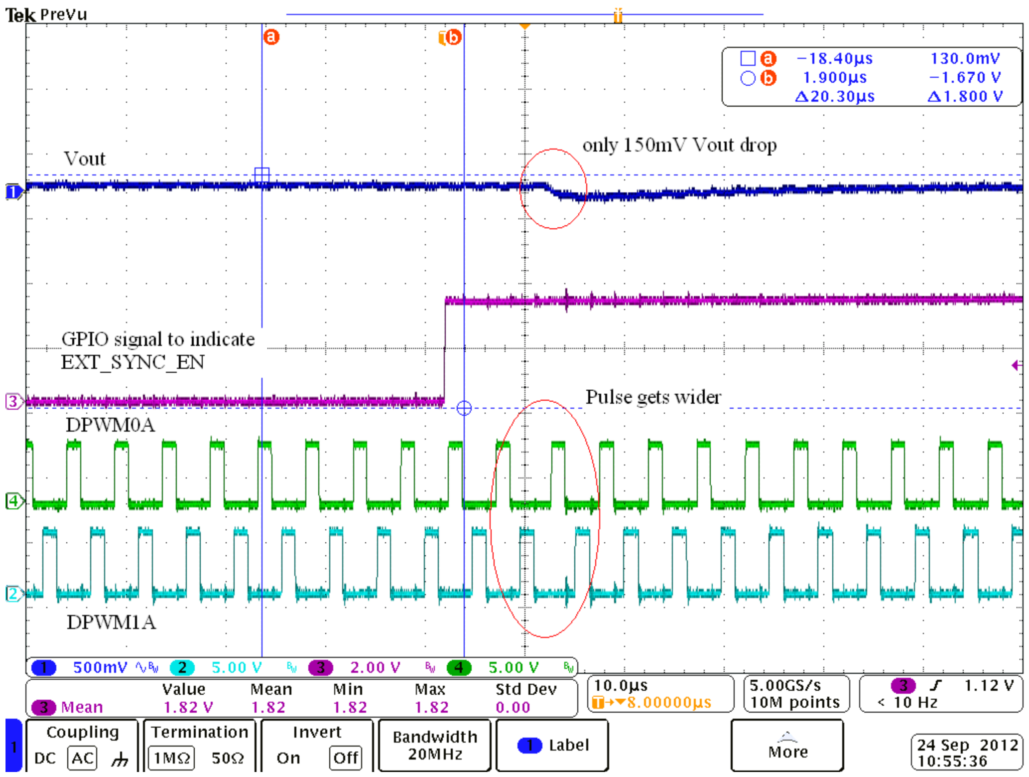
6 Test waveforms

The test is done on UCD3138 hard switching full bridge EVM running at 10A load. The following two figures show the waveform distorted just at the transition point, and the distortion result in only 150mV Vout drop. At other places, the DPWM changes smoothly and Vout is steady and smooth.

The master's frequency is 200kHz. The test on slave is done from switch frequency from 150kHz to 250kHz. In this whole range, the synchronization works fine.



	Value	Mean	Min	Max	Std Dev
3 Mean	2.37 V	2.37	2.37	2.37	0.00


Tek PreVu
only 150mV Vout drop
Vout
GPIO signal to indicate
EXT_SYNC_EN
Pulse gets wider
DPWM0A
DPWM1A
−18.40µs 130.0mV
1.900µs −1.670 V
Δ20.30µs Δ1.800 V
500mV
5.00 V
2.00 V
5.00 V
Value Mean Min Max Std Dev
Mean 1.82 V 1.82 1.82 1.82 0.00
10.0µs
8.00000µs
5.00GS/s
10M points
1.12 V
< 10 Hz
Coupling
DC AC
Termination
1MΩ 50Ω
Invert
On Off
Bandwidth
20MHz
Label
More
24 Sep 2012
10:55:36