Application Report
SLUA729 – October 2014

UCD3138 Integral Branch of the PID Filter

Neil Li
China Telecom Application Team

ABSTRACT

Digital power controller UCD3138 filter is a PID filer, which includes proportion branch, integral branch and differential branch. A register at the output of integrator, named KI_YN, supports write and read operations. In application, writing this register could make the DPWM change rapidly, which is convenient for some designs. The KI_STALL register in the integrator stalls its elements at the current value. It is useful for suppressing response to transients when such response is not desirable. This application note provides the guidance of applying the integrator, as well as several real experimental tests.

Contents

Figures

1 UCD3138 PID Filter and Integral Branch

1.1 UCD3138 PID Filter

Figure 1 shows a block diagram of the first part of the filter, PID architecture. The UCD3138 PID filter includes proportional branch, integral branch, and differential branch. All values shown in the filter are signed numbers. The main input to the filter is a 9-bit signed value from an Error ADC. The main input is shown as XN at the upper-left of Figure 1. The sum of the output of these branches determines the DPWM duty, which is implemented at the cascaded parts of PID filter.

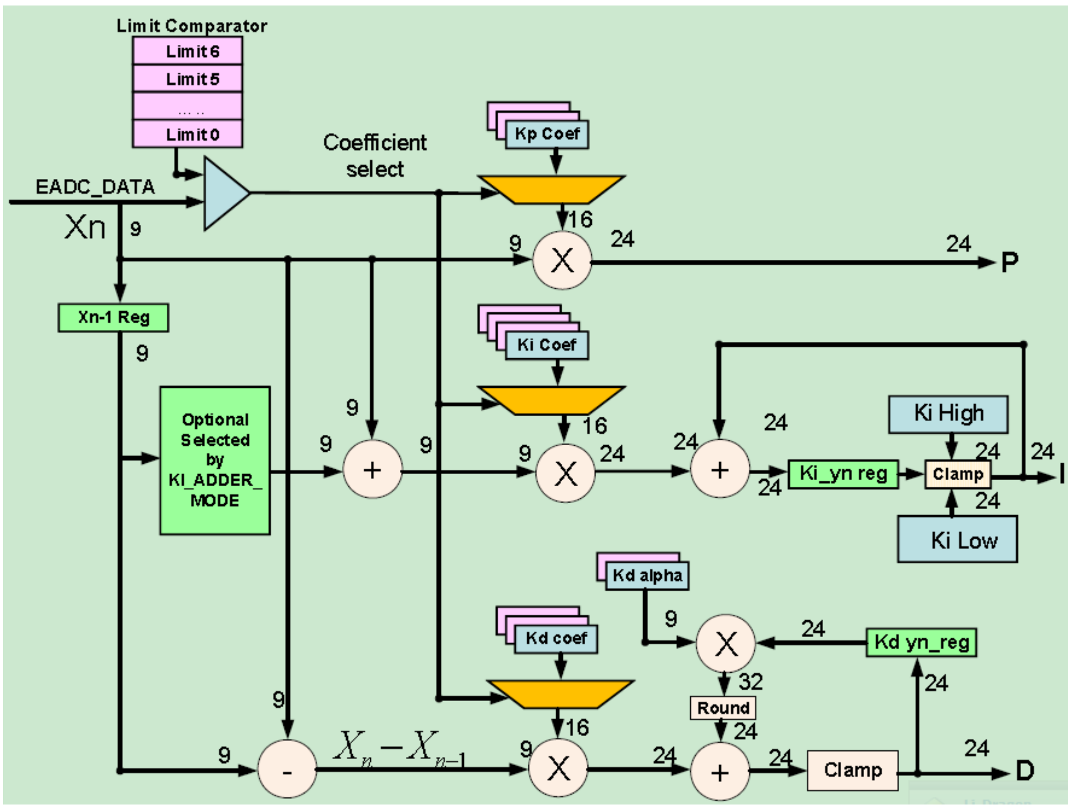

Figure 1. UCD3138 PID Block Diagram

1.2 Integrator Branch

The XN value, or the XN + XN-1 sum is multiplied by the KI coefficient. The output of this multiplication always fits within 24 bits. This value is then added to the existing KI_YN value. The hardware automatically clamps KI_YN value at a 24-bit signed number, and high and low clamp registers (KI_CLAMP_HIGH and KI_CLAMP_LOW) are available which can be used to clamp KI_YN value to lower values. If the value of KI_YN is between KI_CLAMP_HIGH and KI_CLAMP_LOW, the output value of the integrator is exactly the value in KI_YN.

2 The Application of Writing KI_YN

2.1 Change DPWM Duty by Modifying XN

When the power supply is in regulation, the XN is relatively small, hence the output of proportional branch and differential branch are both closed to 0. But, the output of integrator is extremely large. When the XN is modified, the integrator performs calculation and refreshes the output. After a certain period of time, the output is stable with the new value. According to this, the DPWM duty needs a certain time to finish its change. This certain period of time is determined by the coefficients of PID.

Design an experiment to demonstrate. Run open-loop firmware, and change XN from initial 100 to -30. Meanwhile, measure the time that DPWM duty finishes its change. The firmware design follows. Send the PMBus command to trigger the change of XN. DPWM3A is configured to GPIO, used to indicate when the XN is rewritten.

```
Uint8 pmbus_write_ide_config(void)
{
    int i,j,k;

    Filter0Regs.CPUXN.bit.CPU_SAMPLE = -30;
    Dpwm3Regs.DPWMCTRL1.bit.GPIO_A_VAL= 0;
    for (i=0;i<500;i++)
    {
      for (j=0;j<40;j++)
      {
       Dpwm3Regs.DPWMCTRL1.bit.GPIO_B_VAL= 0;
       Dpwm3Regs.DPWMCTRL1.bit.GPIO_B_VAL= 1;
       Dpwm3Regs.DPWMCTRL1.bit.GPIO_B_VAL= 0;
       Dpwm3Regs.DPWMCTRL1.bit.GPIO_B_VAL= 1;
       Dpwm3Regs.DPWMCTRL1.bit.GPIO_B_VAL= 0;
       Dpwm3Regs.DPWMCTRL1.bit.GPIO_B_VAL= 1;
       Dpwm3Regs.DPWMCTRL1.bit.GPIO_B_VAL= 0;
       Dpwm3Regs.DPWMCTRL1.bit.GPIO_B_VAL= 1;
      }
    }
   Dpwm3Regs.DPWMCTRL1.bit.GPIO_A_VAL= 1;

   return PMBUS_SUCCESS;
}
```

Figure 2 shows the waveform. CH1 indicates the DPWM1A, which is controlled by Filter0; CH2 indicates the DPWM3A. After 3.5 ms of XN being written when DPWM3A turns low, the DPWM1A duty finishes its change. This experiment indicates that, using the way of modifying XN, DPWM needs a certain amount of time to complete its change.

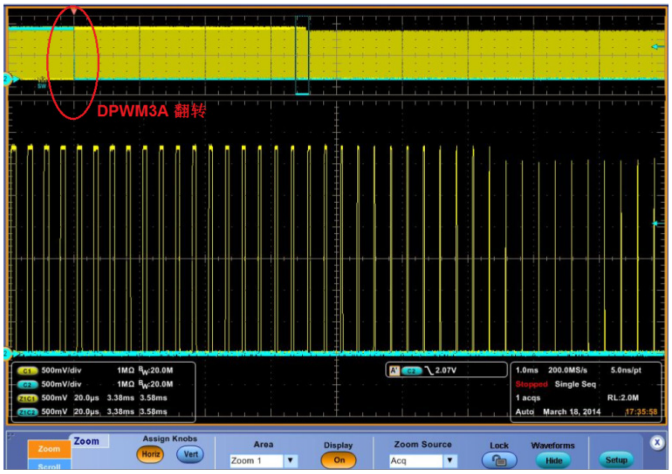

Figure 2. Change DPWM Duty by Modifying XN

2.2 Change DPWM Duty by Modifying KI_YN

Modifying KI_YN can rapidly change the DPWM duty.
Design an experiment to demonstrate. Run open-loop firmware, and write KI_YN directly. The firmware design follows. The value of 0xd99039 is the corresponding value when XN is -30.

```
Uint8 pmbus_write_ide_config(void)
{

    Filter0Regs.CPUXN.bit.CPU_SAMPLE = -30;
    Dpwm3Regs.DPWMCTRL1.bit.GPIO_A_VAL= 0;


    LoopMuxRegs.SAMPTRIGCTRL.bit.FE0_TRIG_DPWM0_EN = 0;
    Filter0Regs.FILTERPRESET.all = (1 << 27) + (1 << 24) + 0xd99039;
    LoopMuxRegs.SAMPTRIGCTRL.bit.FE0_TRIG_DPWM0_EN = 1;


    Dpwm3Regs.DPWMCTRL1.bit.GPIO_A_VAL= 1;

  return PMBUS_SUCCESS;
}
```

Figure 3 shows the waveform. CH1 indicates the DPWM1A, which is controlled by Filter0; CH2 indicates the DPWM3A. After KI_YN is written when DPWM3A turns low, the DPWM1A duty finishes its change rapidly. This experiment indicates that, using the way of modifying KI_YN, DPWM duty can rapidly complete its change.

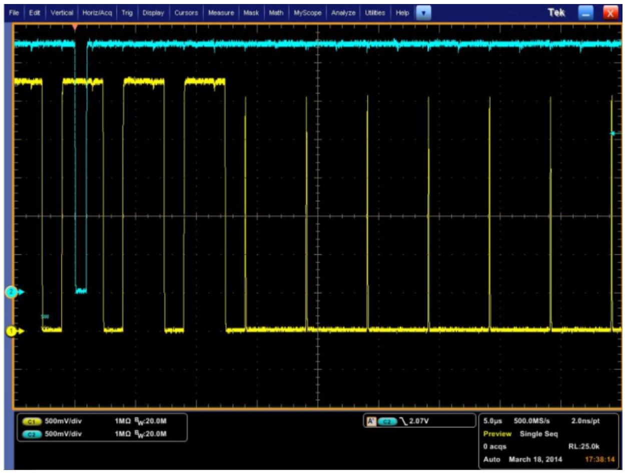

Figure 3. Change DPWM Duty by Modifying KI_YN

2.3 Pre-bias Design

If the output voltage is not 0 V when the power supply with synchronous rectification restarts, the pre-bias is essential. In the power supply based on UCD3138, the pre-bias can design by step and step as below:

1. Acquire the current input voltage and output voltage.
2. Calculate the reference voltage, and then write this value to DAC_VALUE.
3. Calculate the integrator output value with DPWM duty, and then write it into KI_YN.

In the third step, we use the approach described in Section 2.2. The firmware design of the pre-bias function follows.

```
vout = adc_values.Vout;
vdac_prebias = vout * 6.15;  //6.15 is scaler from ADC to DAC
FeCtrl0Regs.EADCDAC.bit.DAC_VALUE = vdac_prebias;

vout =  adc_values.Vout * 6; // output scaler's voltage
vin  =  (vin_eadc_sns  * 30);// input scaler's voltage
duty_prebias = (unsigned int)((((Uint64) vout) << 22) /(Uint64)vin);

LoopMuxRegs.SAMPTRIGCTRL.bit.FE0_TRIG_DPWM0_EN = 0; // Stop triggering EADC2/Filter2
Filter0Regs.FILTERPRESET.all = (1 << 27) + (1 << 24) + preset_value;
LoopMuxRegs.SAMPTRIGCTRL.bit.FE0_TRIG_DPWM0_EN = 1; // Resume triggering EADC2/Filter2
FeCtrl0Regs.RAMPCTRL.bit.FIRMWARE_START = 1;
```

Figure 4 shows the waveform. When the power supply restarts, the output voltage is 2.4 V. After soft-start, the output arrives to its target value, 12 V. During this phase, there is no sharp dip at the output voltage, which indicates the pre-bias works well.

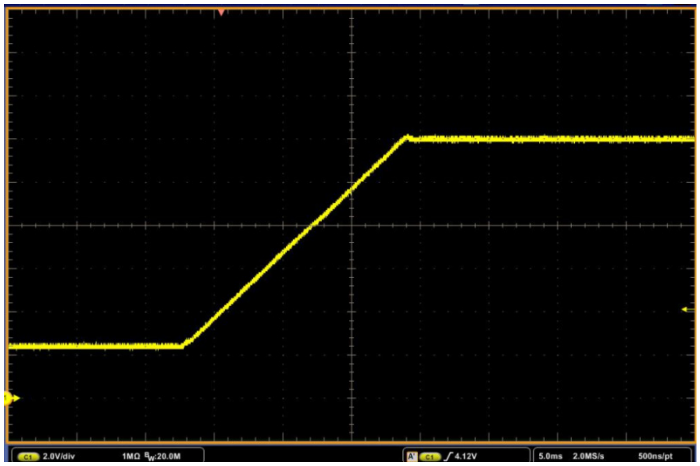

Figure 4. Soft-Start With Pre-bias

3 Stall Calculation of Integral Branch

3.1 Introduction

Stall calculation of integral branch means to freeze the integrator. Therefore, the value in KI_YN is preserved. By setting KI_STALL to 1, this function is enabled, and disabled when KI_STALL is 0. Once disable this function, the integral branch restarts its calculation with the preserved KI_YN value.

3.2 Demonstration

Design an experiment to demonstrate. Run open-loop firmware, and change XN from initial 100 to -30. Before this change, enable the stall by writing 1 into KI_STALL.

```
Uint8 pmbus_write_ide_config(void)
{
    int i,j;
    Filter0Regs.FILTERCTRL.bit.KI_STALL = 1;
    Filter0Regs.CPUXN.bit.CPU_SAMPLE = -30;

    Dpwm3Regs.DPWMCTRL1.bit.GPIO_A_VAL= 0;
    for (i=0; i<500;i++)
    {
      for (j=0; j<40;j++)
      {
       Dpwm3Regs.DPWMCTRL1.bit.GPIO_B_VAL= 0;
       Dpwm3Regs.DPWMCTRL1.bit.GPIO_B_VAL= 1;
       Dpwm3Regs.DPWMCTRL1.bit.GPIO_B_VAL= 0;
       Dpwm3Regs.DPWMCTRL1.bit.GPIO_B_VAL= 1;
       Dpwm3Regs.DPWMCTRL1.bit.GPIO_B_VAL= 0;
       Dpwm3Regs.DPWMCTRL1.bit.GPIO_B_VAL= 1;
       Dpwm3Regs.DPWMCTRL1.bit.GPIO_B_VAL= 0;
       Dpwm3Regs.DPWMCTRL1.bit.GPIO_B_VAL= 1;
      }
    }
    Dpwm3Regs.DPWMCTRL1.bit.GPIO_A_VAL= 1;

  return PMBUS_SUCCESS;
}
```

Figure 5 shows the waveform. CH1 indicates the DPWM1A, which is controlled by Filter0; CH2 indicates the DPWM3A. After about 40 ms of XN being changed, the DPWM1A duty is still mainained. Meanwhile, reading the KI_YN value with Memory Debugger, it shows this value does not change. This unchanged value indicates that after enabling KI_STALL, the KI_YN is, indeed, preserved.

Figure 5. DPWM Duty Change With Enabling Stall

Redesign the firmware, and add the code of setting KI_STALL to 0. This effort intends to check whether or not the integral branch restarts its calculation value with the preserved value before.

```
            }
        }

    Filter0Regs.FILTERCTRL.bit.KI_STALL = 0;
    Dpwm3Regs.DPWMCTRL1.bit.GPIO_A_VAL= 1;

  return PMBUS_SUCCESS;
}
```

Figure 6 shows the waveform. After approximately 3.5 ms of KI_STALL being set 0, when DPWM3A turns high, the duty of DPWM1A changes. This indicates that the integrator restarts its calculation.

Figure 6. Change of DPWM Duty After Disable Stall

4 Conclusion

Writing KI_YN directly, the DPWM duty can be changed rapidly. In some scenarios, this makes the design more convenient, such as the pre-bias mentioned in this document.

Stall calculation of integral branch is a flexibility feature. This feature can be of great assistance to designs which must preserve the KI_YN.

5 References

1. UCD3138 data sheet, Texas Instruments Inc., 2013
2. *UCD3138 Digital Power Peripherals Programmer's Manual*, Texas Instruments Inc., 2013
3. *UCD3138 ARM and Digital System Programmer's Manual*, Texas Instruments Inc., 2013
4. *UCD3138 Monitoring and Communications Programmer's Manual*, Texas Instruments Inc., 2013