UCD31xx

Current Sharing

Application Note

Literature Number: xxxxxx

Date

Table of Contents

1 Current Sharing Overview 3
2 Current sharing module 4
3 Current sharing using average current bus 7
3.1 Average current mode initialization 8
3.2 Average current mode periodic handling 8
4 Current sharing using Master/Slave current bus 11
4.1 Master-Slave mode initialization 12
4.2 Master-Slave mode periodic handling 12
5 Current sharing in droop mode 15

1 Current Sharing Overview

UCD31xx based EVM and reference designs support three major current sharing techniques.

- Current sharing using an average current bus
- Current sharing in Master/Slave mode
- Current sharing in droop mode

Current sharing in all the above mentioned options is handled mostly by firmware and the ARM processor, using several hardware peripherals.

The peripherals that participate in current sharing are:

- ADC-12 to measure both power supply's current and the average bus current

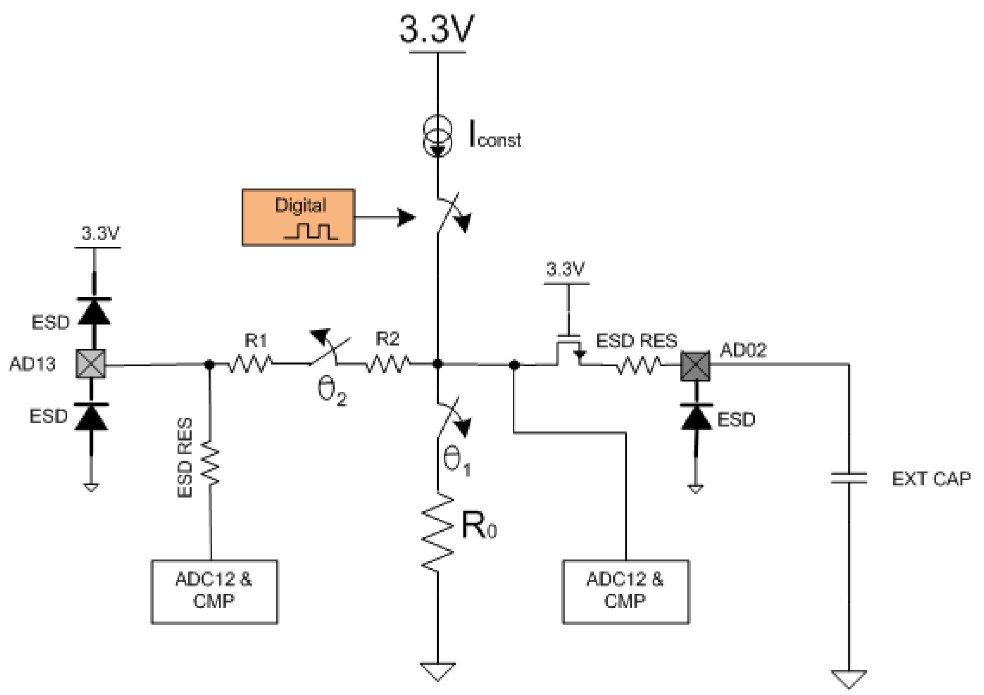
- Current sharing module, to generate the required output for current sharing bus

- EADC's DAC, to slightly trim the output voltage in order for the output current to follow the current sharing bus (Or the voltage droop requirements).

2 Current sharing module

The current sharing module is specially designed to serve the bus driving needs of both average current and the master modes.

Figure 1



Tolerance			
Name	Typical	+- %	Comment
R0	10K	2	Accurate internal resistor with temp. compensation
R1	400	25	Resistor with switch resistance lump
R2	3.2K	25	Resistor support no more than 1mA
ESDRES	250	25	ESD resistor
Iconst	250uA	2	Use in PWM output current

Current Sharing Mode	CSCTRL.bit.TEST_MODE	Switch Ө1	Switch Ө2	Digital - PWM
Tri-state or Slave mode	0	OFF	OFF	OFF
PWM average current Bus	1	ON	OFF	ACTIVE
Analog average current Bus or Master mode	3	OFF	ON	OFF

Important notes (Naming Caveats):

- The field TEST_MODE inside the CSCTRL (Current Sharing Control register) is the bit field that determines the current sharing operation mode.

- The field DPWM_PERIOD inside the CSCTRL (Current Sharing Control register) is the bit field that determines the frequency of current sharing PWM, and has nothing to do with the DPWM engines that run at switching frequency.

Generating an output signal that its voltage is proportional to the current driven by the output of the power supply can be done in one of the following ways.

- Output current is sensed by an accurate and stable shunt resistor. In this case an analog signal conditioning circuit can scale (amplify) the current sensing signal and provide the bus with the required signal. In this case the signal needs to be connected to the AD13 input pin.

  If current sharing mode 3 is chosen, the switch Ө2 is closed and the signal connected to AD13 will propagate through the switch and drive the bus via the AD02 pin.

- Output current is sensed on a PCB copper trace; therefore it needs to go through temperature compensation and calibration. In this case ADC may be used to provide digital signal to ARM processor and firmware will handle the temperature compensation and calibration. The digital signal then will be converted back to analog using a PWM mechanism available at UCD31xx's "current sharing peripheral".

  If current sharing mode 1 is chosen, then the PWM current source is active, and the switch Ө1 is closed. Therefore the voltage generated by the PWM current flowing through the resistor R0 will drive the bus via the AD02 pin.

Please note:

- The AD02 pin is carefully designed so when the device is not powered it will not pull down the current sharing bus. This is essential for the proper operation of current sharing mechanism, when one of the power supply units looses its power.

- An external passive low pass filter in between the current sharing bus and the ADC02 pin will help to improve the ADC reading accuracy. This in turn will improve the accuracy of current sharing effort. An RC filter of R=220 Ohms and C=10nF were used in HSFB (Hard Switching Full Bridge) EVM by Texas Instruments.

The value of the external capacitor on current share bus needs to be calculated based on the following requirements:

1. Whether the internal PWM is used for current sharing.
2. The amount of switching noise coupled/injected into the bus in a specific board layout.
3. The desired current sharing's control loop bandwidth.
4. The desired level of current sharing accuracy which will dictate:
   4-1) The frequency of the internal PWM operation.
   4-2) The acceptable level of ripple on the bus.

When the internal PWM is running with a 200 KHZ frequency, a capacitor of Cext = 0.033 µF will offer a 10mV peak to peak ripple level.

The practical value of the capacitor should be determined by the application based on the above criteria.

3 Current sharing using average current bus

This technique relies on the voltage on the shared bus to be proportional to the average current supplied by all the power supplies in the system.

Each power supply provides an output signal that is proportional to the current that it drives at its output.

All above mentioned output signals have the same output impedance and all connect to the same current sharing bus.

Therefore the voltage at the bus will be the average voltage of all these signals.

Each power supply is attempting to adjust its output current in order to match its output current with the average current output of all the power supplies.

The output current of each power supply and the average current of all power supplies are both measured by ADC. Then through a current sharing algorithm the value of the output voltage is slightly changed in order to make the output current match the average current.

Since the level of the regulated output voltage is dictated by the EADC's DAC value, the firmware will feed (change) this value periodically in order to keep matching the average current.

Since the output impedance of a power supply can be as low as a fraction of milli-Ohm; a very small change in the output voltage will cause a substantial change in the output current. Therefore the DAC dithering feature needs to be turned on.
We need the entire 14 bits of EADC-DAC resolution to execute the current sharing functionality, and the last 4 bits are only available thanks to the dithering.

Please note: For those application where the output impedance of power-supply is high enough, output ripple may be reduced by turning the dithering off (recommended).

3.1 Average current mode initialization

```
// Enable EADC's DAC dithering in order to increase the resolution to 14 bits
FeCtrl0Regs.EADCDAC.bit.DAC_DITHER_EN = 1;
```

If average current bus driven by analog bus output is chosen, use:

```
// Set the current sharing bus output to pass through, switches Θ1=open,
Θ2=close
MiscAnalogRegs.CSCTRL.bit.TEST_MODE = 3;
```

If average current bus driven by digital PWM bus output is chosen, use:

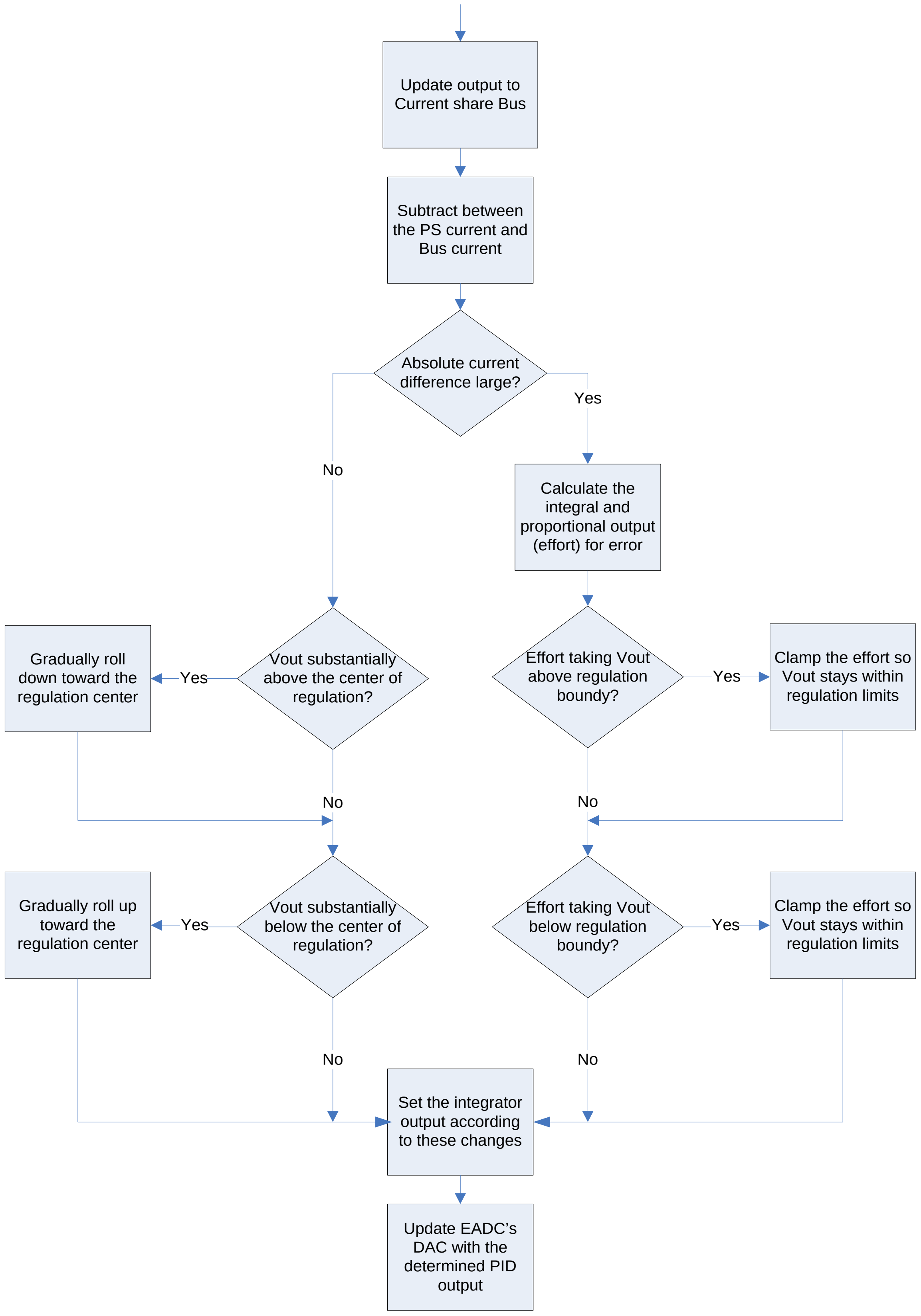
```
// Set the current sharing bus output to PWM mode, switches Θ1=close, Θ2=open
MiscAnalogRegs.CSCTRL.bit.TEST_MODE = 1;

// Set the current sharing pwm frequency as desired
// when the LSB resolution is 31 ns (nano-seconds)
MiscAnalogRegs.CSCTRL.bit.DPWM_PERIOD = desired_period; // 108 -> 300 KHZ
```

The ADC-12 also needs to be configured properly so the channels AD13 and AD02 will measure the output current and the bus average current respectively.

3.2 Average current mode periodic handling

The function `void handle_current_sharing_average(void)` is called every 100 µs from the standard interrupt and only when the state machine is in regulated state. Therefore active current sharing is executed only during regulation at the final output voltage setpoint.



Update output to Current share Bus
Subtract between the PS current and Bus current
Absolute current difference large?
Yes
No
Calculate the integral and proportional output (effort) for error
Gradually roll down toward the regulation center
Yes
Vout substantially above the center of regulation?
Effort taking Vout above regulation boundy?
Yes
Clamp the effort so Vout stays within regulation limits
No
No
Gradually roll up toward the regulation center
Yes
Vout substantially below the center of regulation?
Effort taking Vout below regulation boundy?
Yes
Clamp the effort so Vout stays within regulation limits
No
No
Set the integrator output according to these changes
Update EADC's DAC with the determined PID output

Figure 2

Below is the pseudo code for `void handle_current_sharing_average(void)` functionality, for the real code please refer to “UCD3138LLCEVM” reference code.

```
void handle_current_sharing_average(void)
{

  // Update the current sharing PWM with the power supply’s new output
  // current. This line of the code needed only if the digital pwm output
  // generation for current sharing bus is selected
  MiscAnalogRegs.CSCTRL.bit.DPWM_DUTY = adc_values_avg.io_sense * scaling_factor;

  //Calculate the difference between average current on the bus and output current
  local_error = adc_values_avg.ishare - adc_values_avg.io_sense;

  //if the absolute value of current difference is large enough to require an action
  if(abs(local_error) > ishare_threshold)
  {
     // calculate the integral element of PID control based on current difference
    current_share_int_state = current_share_int_state + current_share_ki * local_error;
    // calculate the proportional element of PID and add on the top of integral part
    current_share_control_effort = current_share_int_state + current_share_kp * local_error;

    //if result is trying to change the output voltage over the regulation limit
     //note: current_share_control_effort is shifted by (CS_INT_EXP = 10) in order to
     // to gain fixed arithmetic calculation resolution
     if( current_share_control_effort >  (eadc_dac_max << CS_INT_EXP))
     {
       //Clamp the current share effort so the output voltage will not go too high
       current_share_control_effort = (eadc_dac_max << CS_INT_EXP);
       current_share_int_state = current_share_control_effort;
     }
     if result is trying to change the output voltage below the regulation limit
     else if(current_share_control_effort < (eadc_dac_min << CS_INT_EXP))
     {
       //Clamp the current share effort so the output voltage will not go too low
       current_share_control_effort = (eadc_dac_min << CS_INT_EXP);
       current_share_int_state = current_share_control_effort;
     }
    //if current difference small enough to require no current sharing action
    //edge the output voltage gently back toward center of voltage regulation range
    else
    {
       //if vout was previously forced substantially above center of voltage regulation
       if(((current_share_control_effort >> CS_INT_EXP) - eadc_dac_target) > limit)
       {
         // reduce the voltage gradually toward the center of voltage regulation range
         current_share_control_effort = current_share_control_effort - 16;
       }
       //if vout was previously forced substantially below center of voltage regulation
      else if(((current_share_control_effort >> CS_INT_EXP) - eadc_dac_target) < -limit)
       {
       //increase the voltage gradually toward the center of voltage regulation range
         current_share_control_effort = current_share_control_effort + 16;
       }
       //set the integrator state with the updated value for the next cycle
       current_share_int_state = current_share_control_effort;
    }

       //shift the PID result back to the physical resolution and update the
      // EADC’s DAC value with all 14 bits.
       FeCtrl0Regs.EADCDAC.bit.DAC_VALUE = current_share_control_effort >> CS_INT_EXP;
}
```

4 Current sharing using Master/Slave current bus

This technique relies on the voltage on the shared bus to be driven only by the master.

All power supplies are set into Master-mode immediately after power up. But only one unit stays Master in steady state.

In steady state, all other power supplies in the system will be Slaves, and will attempt to match the current dictated by the Master of the bus.

Master is defined as the power supply that shares the highest current to the output compared to the rest of the power supplies.

If the master power supply is removed, then the next power supply with the highest current becomes the master. Each Slave will be able to become master is its output current is above the current indicated by the shared bus.

As previously mentioned, all power supplies are set into Master-mode immediately after power up. This ensures that the bus driven at least by one power supply at all times.

It is imperative that the bus is always driven by at least one power supply; otherwise bus could be in Tri-state mode, therefore having a high voltage. In this case all power supplies could assume that they are slaves and a master is driving a higher current. This would lead to a dead lock, where no power supply will become a master.

A pull-up resistor connected to the current bus will also be helpful to prevent the above scenario.

The output current of each slave power supply and the bus current (Master's current) are both measured by ADC. Then through a current sharing algorithm the value of the output voltage is slightly changed in order to make the output current match the bus current.

Since the level of the regulated output voltage is dictated by the EADC's DAC value, the firmware will feed (change) this value periodically in order to keep matching the Master's current.

Since the output impedance of a power supply can be as low as a fraction of milli-Ohm; a very small change in the output voltage will cause a substantial change in the output current. Therefore the DAC dithering feature needs to be turned on.
We need the entire 14 bits of EADC-DAC resolution to execute the current sharing functionality, and the last 4 bits are only available thanks to the dithering.

Please note: For those application where the output impedance of power-supply is high enough, output ripple may be reduced by turning the dithering off (recommended).

4.1 Master-Slave mode initialization

```
// Enable EADC's DAC dithering in order to increase the resolution to 14 bits
FeCtrl0Regs.EADCDAC.bit.DAC_DITHER_EN = 1;


// Set the current sharing pwm frequency as desired
// when the LSB resolution is 31 ns (nano-seconds)
MiscAnalogRegs.CSCTRL.bit.DPWM_PERIOD = desired_period; // 108 -> 300 KHZ
```

The ADC-12 also needs to be configured properly so the channels ADC-13 and ADC-02 will measure the output current and the bus average current respectively.

4.2 Master-Slave mode periodic handling

The function `void handle_current_sharing_master_slave (void)` is called every 100 µs from the standard interrupt and only when the state machine is in regulated state. Therefore active current sharing is executed only during regulation at the final output voltage setpoint.

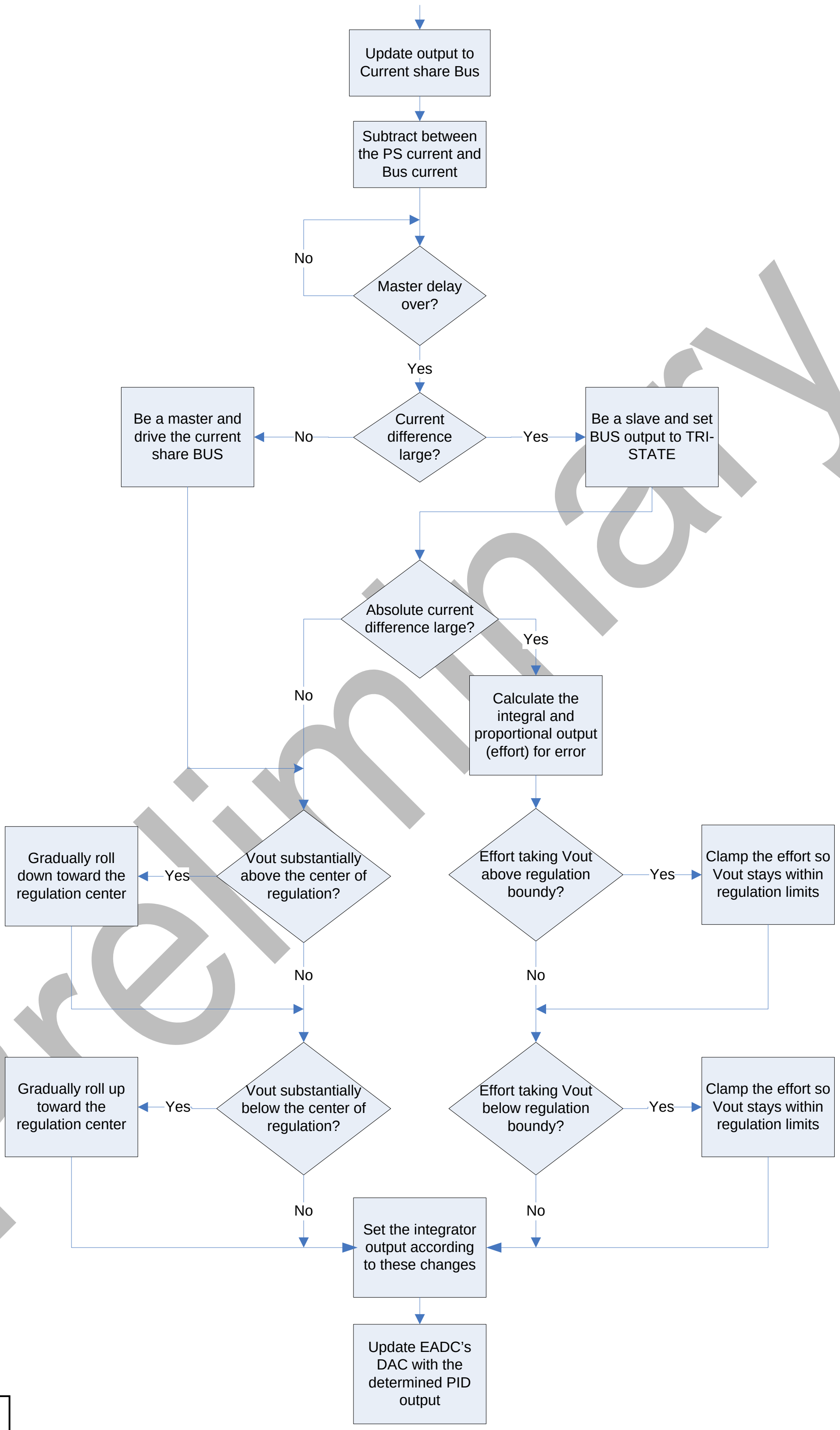

Update output to Current share Bus
Subtract between the PS current and Bus current
No
Master delay over?
Yes
Be a master and drive the current share BUS
No
Current difference large?
Yes
Be a slave and set BUS output to TRI-STATE
Absolute current difference large?
Yes
No
Calculate the integral and proportional output (effort) for error
Gradually roll down toward the regulation center
Yes
Vout substantially above the center of regulation?
Effort taking Vout above regulation boundy?
Yes
Clamp the effort so Vout stays within regulation limits
No
No
Gradually roll up toward the regulation center
Yes
Vout substantially below the center of regulation?
Effort taking Vout below regulation boundy?
Yes
Clamp the effort so Vout stays within regulation limits
No
No
Set the integrator output according to these changes
Update EADC's DAC with the determined PID output

Figure-3

Below is the pseudo code for `void handle_current_sharing_master_slave (void)` functionality, for the real code please refer to "UCD3138LLCEVM" reference code.

```
void handle_current_sharing_master_slave(void)
{
  // Update the current sharing PWM with the power supply's new output
  // current. This line of the code needed only if the digital pwm output
  // generation for current sharing bus is selected
  MiscAnalogRegs.CSCTRL.bit.DPWM_DUTY = adc_values_avg.io_sense * scaling_factor;

  //Calculate the difference between average current on the bus and output current
  local_error = adc_values_avg.ishare - adc_values_avg.io_sense;

  //Wait unit master_time_count is equal to master_time_limit before current sharing
  if (master_time_count < master_time_limit)
  {
       master_time_count++;
       master_state = 1;
       // Set the current sharing bus output to PWM mode, switches Θ1=close, Θ2=open
       // if master drives bus using an originally analog signal then replace 1 with 3
       MiscAnalogRegs.CSCTRL.bit.TEST_MODE = 1; // or =3; if analog signal used
  }
  else
  {
       //if self current is larger than bus current, be the master
       if (local_error < ishare_threshold_master_enable)
       {
              master_state = 1;
              // Set the current sharing bus output to PWM mode, switches Θ1=close, Θ2=open
              // if master drives bus using an originally analog signal then replace 1 with 3
              MiscAnalogRegs.CSCTRL.bit.TEST_MODE = 1; // or =3; if analog signal used
       }
       //if self current is not larger than bus current, be the slave.
       else if (local_error > ishare_threshold_slave_enable)
       {
              master_state = 0;
              // Set the current output mode to slave(Tri-state) mode,
              // switches Θ1=close, Θ2=open
              MiscAnalogRegs.CSCTRL.bit.TEST_MODE = 0;
       }
  }
  //if the absolute value of current difference is large enough to require an action
  // and if not already a master
  if((abs(local_error) > ishare_threshold_ms) & (!master_state))
  {
       // calculate the integral element of PID control based on current difference
       current_share_int_state = current_share_int_state + current_share_ki * local_error;
       // calculate the proportional element of PID and add on the top of integral part
       current_share_control_effort = current_share_int_state + current_share_kp * local_error;
       //if result is trying to change the output voltage over the regulation limit
       //note: current_share_control_effort is shifted by (CS_INT_EXP = 10) in order to
       // to gain fixed arithmetic calculation resolution
       if(current_share_control_effort >  (eadc_dac_max << CS_INT_EXP))
       {
          //Clamp the current share effort so the output voltage will not go too high
          current_share_control_effort = (eadc_dac_max << CS_INT_EXP);
            current_share_int_state = current_share_control_effort;
       }
       if result is trying to change the output voltage below the regulation limit
       else if(current_share_control_effort < (eadc_dac_target << CS_INT_EXP))
       {
          //Clamp the current share effort so the output voltage will not go too low
          current_share_control_effort = (eadc_dac_target << CS_INT_EXP);
          current_share_int_state = current_share_control_effort;
       }
  }
  //if current difference small enough to require no current sharing action
  //edge the output voltage gently back toward center of voltage regulation range
   else if (master_state)  {
      //if vout was previously forced substantially above center of voltage regulation
      if(((current_share_control_effort >> CS_INT_EXP)  - eadc_dac_target) > limit)
      {
         // reduce the voltage gradually toward the center of voltage regulation range
         current_share_control_effort = current_share_control_effort - 16;
      }
      //if vout was previously forced substantially below center of voltage regulation
      else if(((current_share_control_effort >> CS_INT_EXP) - eadc_dac_target) < -limit)
      {
         //increase the voltage gradually toward the center of voltage regulation range
         current_share_control_effort = current_share_control_effort + 16;
      }
      //set the integrator state with the updated value for the next cycle
      current_share_int_state = current_share_control_effort;
   }
    //shift the PID result back to the physical resolution and update the
    // EADC's DAC value with all 14 bits
    FeCtrl0Regs.EADCDAC.bit.DAC_VALUE = (current_share_control_effort >> CS_INT_EXP);
 }
```

5 Current sharing in droop mode

Droop mode current sharing does not require any shared bus.
The current sharing is executed by simulating a resistive behavior at the output terminals.

Firmware will initiate an output current measurement via an ADC and will slightly reduce the output voltage if output current increases.

The reduction of output voltage will in turn cause a reduction in current shared by the specific power supply and this change acts like a slow negative feedback that regulates the amount of current contributed to shared output junction.

The amount of output voltage change as a response to output current change is referred to as voltage droop slope. The voltage droop slope is configurable and need to be set as a function of the power supply's output resistance up to the shared output junction.

This output voltage drop resembles existence of a physical resistance in series to the output terminals, but it has no adverse effect on response to load transients.

When use of any hardware resistance will always degrade the output load transient response; the digital voltage droop current sharing is quite slow and does not increase the output impedance of the power supply in non-DC (higher frequency) domain.

The implementation of droop current sharing is relatively straight forward and will not be discussed here in details.

Please note that current sharing in droop mode requires high levels of initial output voltage setting accuracies. These levels of accuracy may in turn require output voltage calibration.