Floating Load Buck Regulator Datasheet FLBUCK V 1.1

Copyright © 2009-2012 Cypress Semiconductor Corporation. All Rights Reserved.

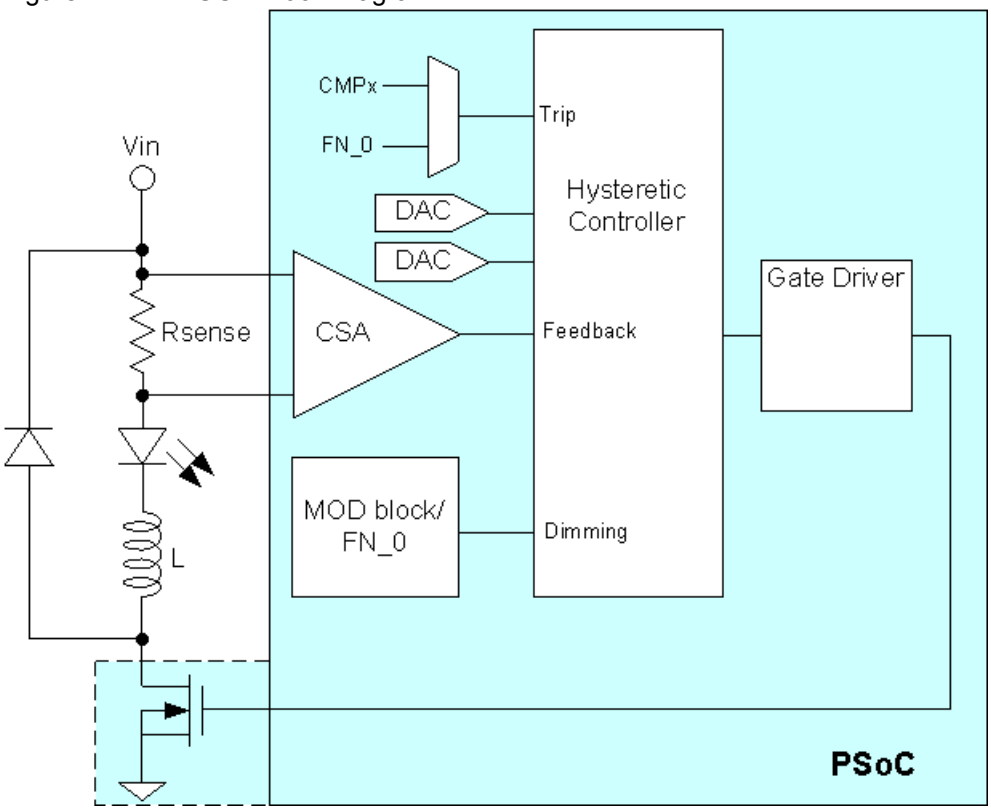
Resources	PSoC® Blocks			API Memory (Bytes)		Pins (per External I/O)
	CSA	MOD	HYSTCTL+ FET	Flash	RAM	
CY8CLED0xD, CY8CLED0xG	1	1	1	480	6	3

Features and Overview

- Highly efficient hysteretic current-mode buck converter
- 50 kHz to 2 MHz effective switching frequency range
- Ideal topology for LED applications
- 4 independent channels
- Independent configurable linear and PWM dimming
- Optional current or temperature protection

The Floating Load Buck Regulator User Module (FLBUCK) allows you to create a fully functional LED driver with dimming and temperature protection based on a current mode buck converter. It also can be used as a general purpose buck converter that regulates the output current, not the voltage.

Figure 1. FLBUCK Block Diagram



Functional Description

The Floating Load Buck Regulator User Module is intended for LED applications as a complete current regulator. It provides current stabilization when output voltage is less than input voltage. The Floating Load Buck User Module can accept a maximum of 36V. Current monitoring is carried out with RSENSE voltage drop tracking. It is proportional to the current that is running through the load.

An additional feature of the buck regulator is output signal dimming modulation. The FLBUCK User Module performs one of the three kinds of modulation and provides overcurrent protection.

The FLBUCK UM consolidates several other modules such as the Current Sense Amplifier (CURSENSEHW), the Hysteretic Controller (HYSTCTRL) and one of the modulator user modules (PWM16HW, DMM16HW, or PRISM16HW) into a single-channel closed-loop buck converter. Refer to the corresponding user module datasheets for details about how each of these user modules function.

The current sense amplifier (CSA) produces a feedback voltage that is proportional to the current through Rsense. The voltage range of the DACs in the Hysteretic Controller are configurable to either 1.3V or 2.6V. The hysteretic controller (HCT) compares this feedback value to the upper and lower thresholds. Going below the lower threshold turns the switch ON and exceeding the upper threshold turns the switch OFF. The average load current is thus kept near the predefined level. The gate driver strength may be configured when an internal FET is used.

The MOD block performs a modulation function. The FLBUCK User Module has three modulation mechanisms:

- Pulse-width modulation (PWM mode)
- Density modulation (DMM mode)
- Stochastic signal density modulation (PrISM™ mode)

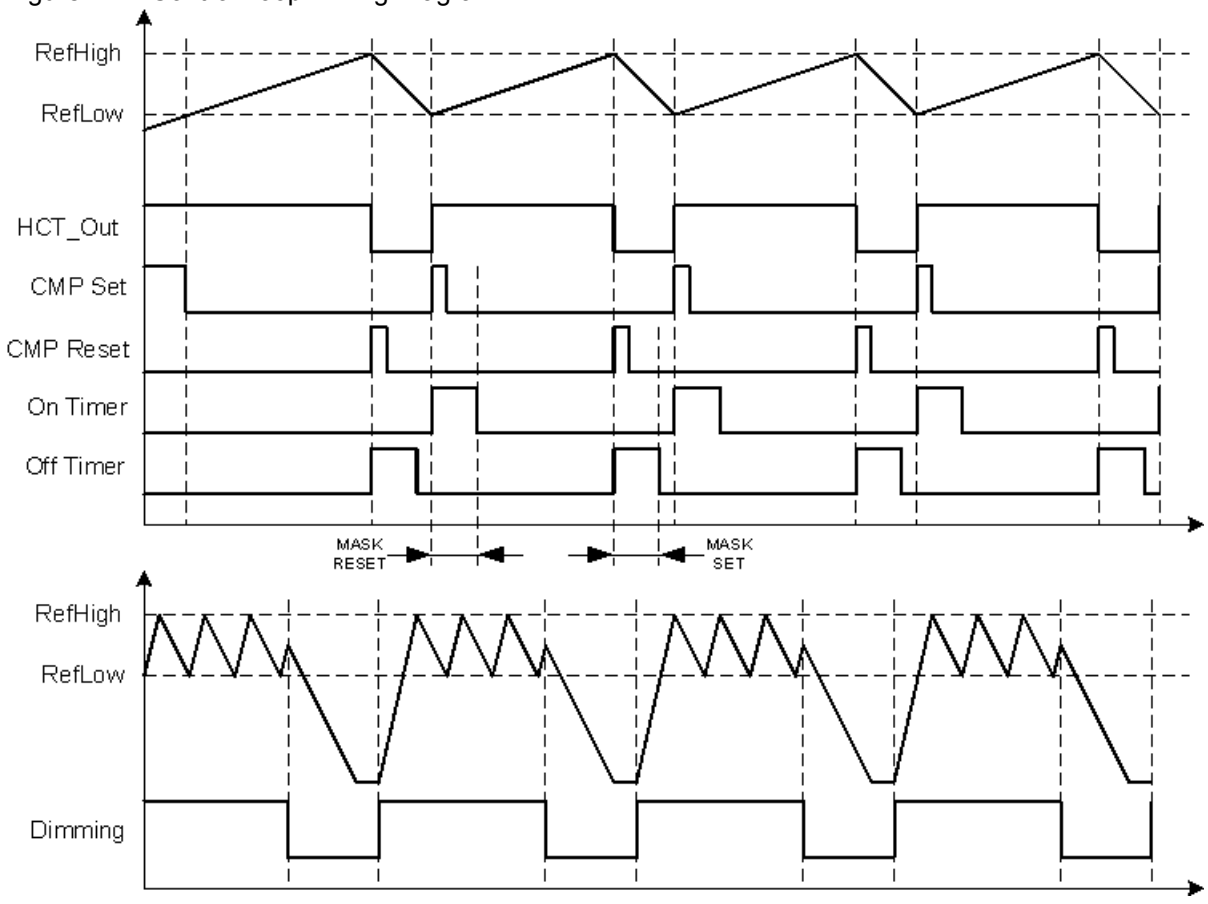
Refer to the corresponding user module datasheets for details about how each of these user modules function.

The FLBUCK output signal is created by ANDing the hysteretic controller main loop, the modulation function, and the signal from the overcurrent protection circuit.

The trip function notifies the buck convertor if a current value exceeds the threshold. The trip function, like modulation functions, overrides the normal hysteretic controller main loop operation.

The following timing diagram shows the buck converter control loop and dimming function.

Figure 2. Control Loop Timing Diagram



When the switch is on the current through coil ramps up:

Equation 1

$$i(t) = i_0 + \frac{V_{in} - V_{out}}{L} \cdot t$$

When the switch is off the current through coil ramps down:

Equation 2

$$i(t) = i_1 + \frac{V_{out}}{L} \cdot t$$

Assuming the gain of the current sense amplifier is G_{CSA} the feedback voltage is:

Equation 3

$$V(t) = G_{CSA} \cdot R_{sense} \cdot i(t)$$

$$V(t) = G_{CSA} \cdot R_{sense} \cdot i_0 + \left(\frac{V_{in} - V_{out}}{L} \cdot t \right), \text{switch is On}$$

$$V(t) = G_{CSA} \cdot R_{sense} \cdot \left(i_1 + \frac{V_{out}}{L} \cdot t \right), \text{switch is Off}$$

In the steady state:

Equation 4

$$V(t) = REF_B + G_{CSA} \cdot R_{sense} \cdot \frac{V_{in} - V_{out}}{L} \cdot t, \text{switch is On}$$

$$V(t) = REF_A + G_{CSA} \cdot R_{sense} \cdot \frac{V_{out}}{L} \cdot t, \text{switch is Off}$$

These equations allow you to derive the main equations that describe how the buck converter works. For example, the switching frequency is:

Equation 5

$$F_{sw} = \frac{G_{CSA} \cdot R_{sense} \cdot (V_{in} - V_{out}) \cdot V_{out}}{(REF_A - REF_B) \cdot L \cdot V_{in}}$$

The duration of the switch on state is:

Equation 6

$$T_{on} = \frac{(REF_A - REF_B) \cdot L}{G_{CSA} \cdot R_{sense} \cdot (V_{in} - V_{out})}$$

The duration of the switch off state is:

Equation 7

$$T_{off} = \frac{(REF_A - REF_B) \cdot L}{G_{CSA} \cdot R_{sense} \cdot V_{out}}$$

The duty cycle is:

Equation 8

$$D = \frac{V_{out}}{V_{in}}$$

The average load current is:

Equation 9

$$I_{out} = \frac{REF_A + REF_B}{2 \cdot G_{CSA} \cdot R_{sense}}$$

The current ripple, pp:

Equation 10

$$I_{ripple} = \frac{REF_A - REF_B}{G_{CSA} \cdot R_{sense}}$$

The peak current through coil:

Equation 11

$$I_{peak} = \frac{REF_A}{G_{CSA} \cdot R_{sense}}$$

DC and AC Electrical Characteristics

See the device characterization data in the DC and AC Electrical Characteristics section of the device datasheet.

Placement

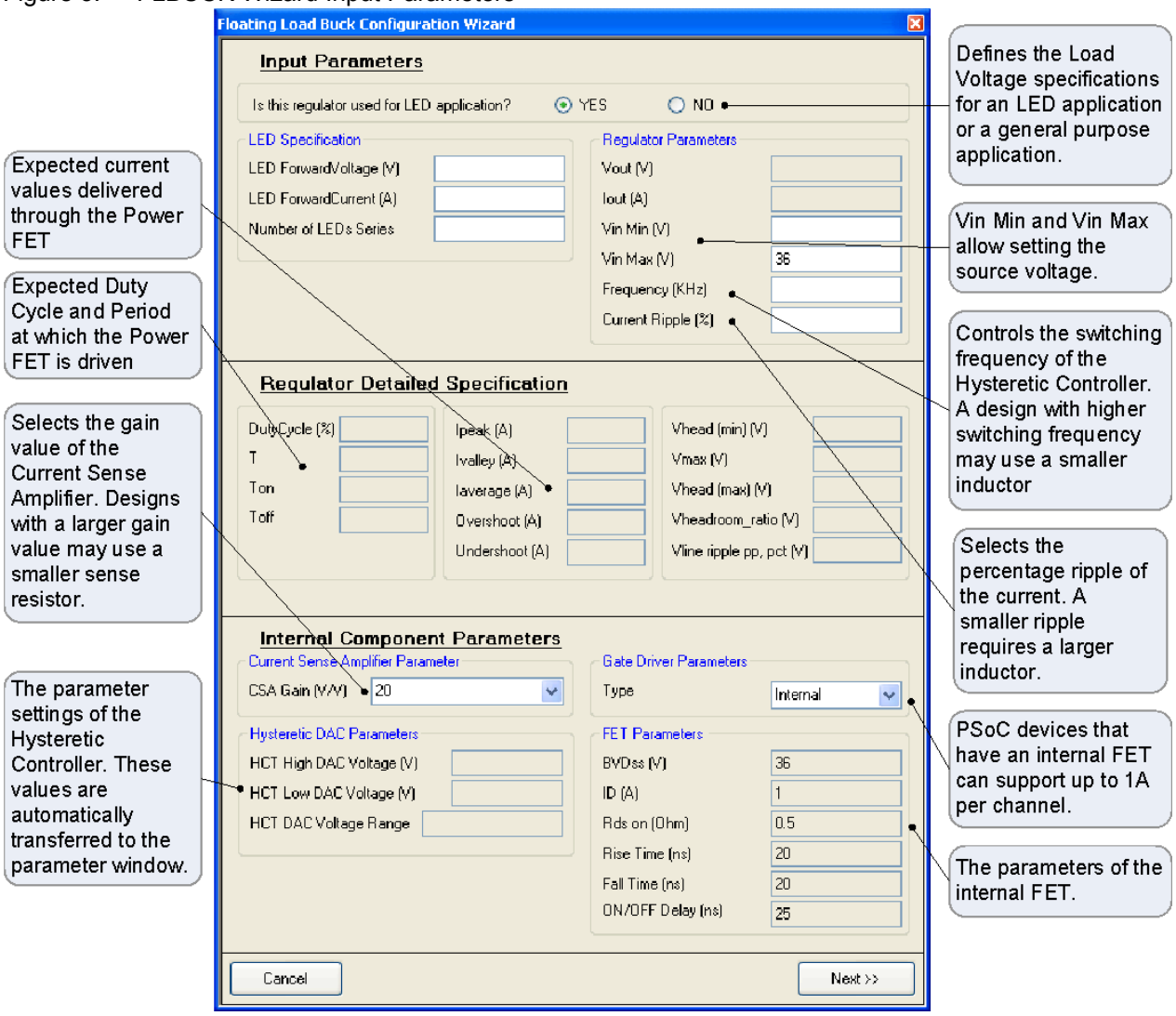
The Floating Load Buck User Module occupies a combination of corresponding power blocks (CSA, MOD, HYSTCTRL, and FET).

Wizard

To access the Wizard, right click the FLBUCK User Module in the Workspace Explorer, then select the **FLBUCK Wizard** with a left mouse click.

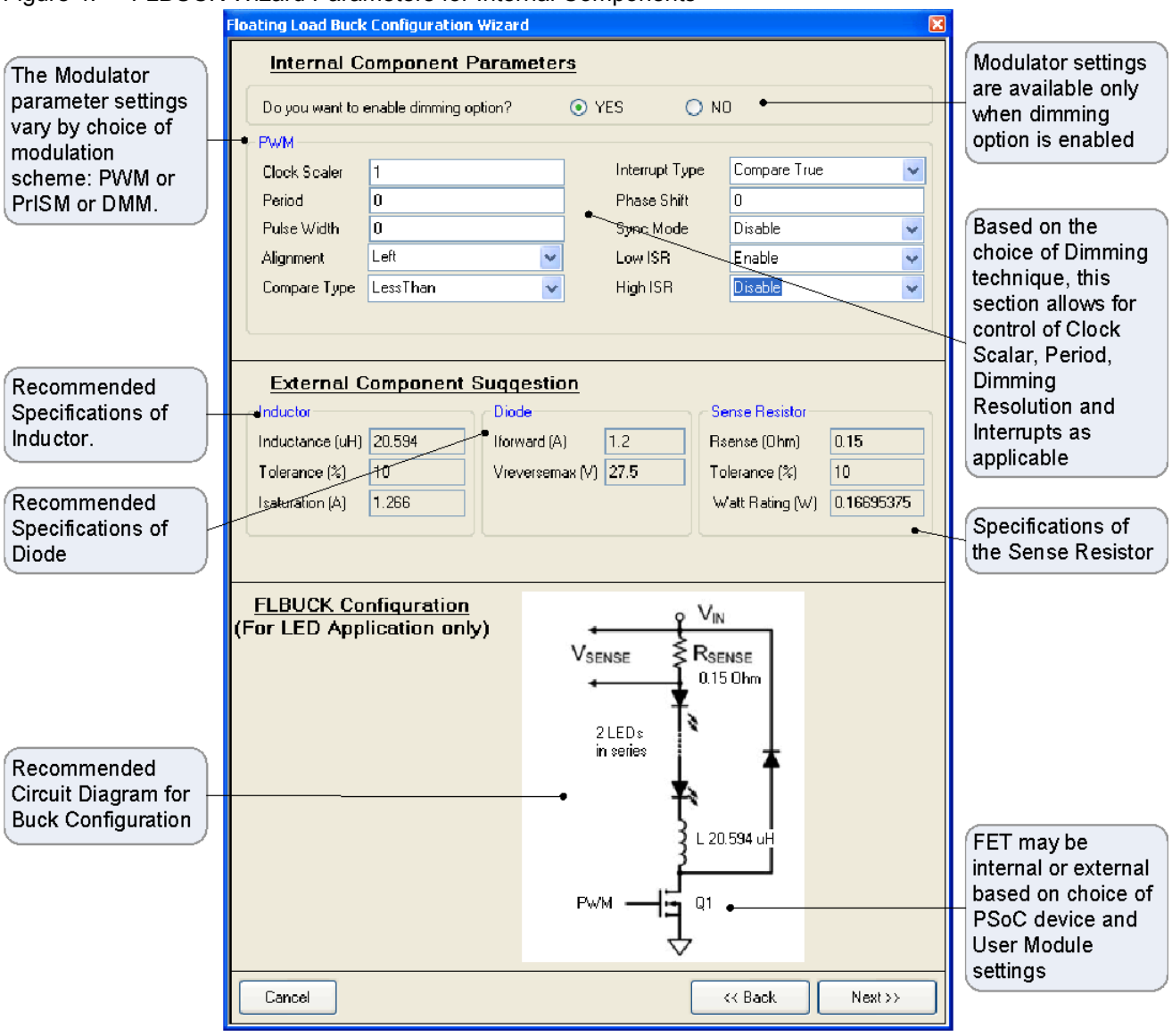
The wizard automatically updates the values of the associated User Module parameters.

Figure 3. FLBUCK Wizard Input Parameters



The next page of the wizard covers MOD block parameters based on the configuration type (PWM, DMM or PRISM). The middle section is informative.

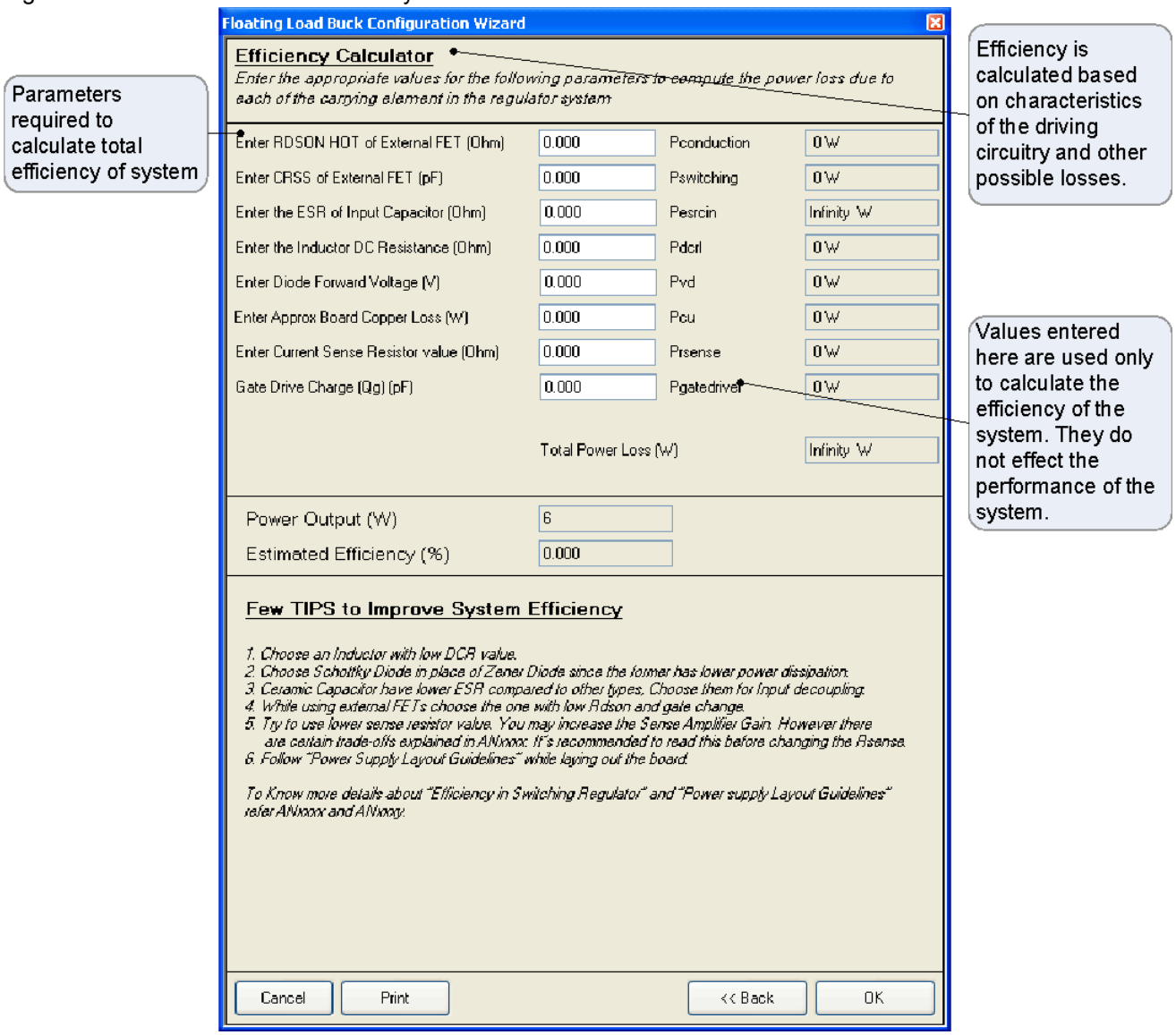
Figure 4. FLBUCK Wizard Parameters for Internal Components



The last page is efficiency calculator. It needs several characteristics to estimate an approximate power loss.

Characteristic	Unit	Description
RDSON	Ohms	The hot on-resistance of the MOSFET
CRSS	pF	Reverse-transfer capacitance of high-side MOSFET
ESR	Ohms	Equivalent series resistance of input capacitor
Inductor DC Resistance	Ohms	Equivalent series resistance of inductor
Diode Forward Voltage	V	Diode voltage drop
Approx Board Copper Loss	W	Approximate PC-board copper loss
Current Sense Resistance	Ohms	Current sense resistance
Gate Driver Charge	Qg	Gate driver charge

Figure 5. FLBUCK Wizard Efficiency Calculator



Parameters and Resources

The following table summarizes all parameters and classifies them by configuration. Current sense amplifier block, modulation block, and hysteretic controller block names are abbreviated to CSA, MOD and HCT.

FLBUCK function	PWM	DMM	PRISM
CSA Gain	●	●	●
CSA Bandwidth	●	●	●
MOD ClockScaler	●	●	●
MOD Period	●	●	–
MOD DimmingResolution	–	–	●
MOD PulseWidth	●	–	–
MOD SignalDensity	–	●	●
MOD ModResolution	–	●	–
MOD CompareType	●	–	●
MOD Alignment	●	●	–
MOD SyncMode	●	●	–
MOD FrequencyCompensation	–	–	●
MOD InterruptType	●	●	–
MOD PhaseShift	●	●	–
MOD LowISR	●	●	–
MOD HighISR	●	●	–
HCT RefHigh	●	●	●
HCT RefLow	●	●	●
Gate Driver	●	●	●
HCT GateDriverStrength	●	●	●
HCT FeedbackInput	●	●	●

FLBUCK function	PWM	DMM	PRISM
HCT DACVoltageRange	●	●	●
HCT DimInput	●	●	●
HCT TripInput	●	●	●
HCT TimerDelay	●	●	●

CSA Gain

This parameter sets the current sense amplifier gain. It controls both Stage 1 gain and Stage 2 bypass. The following options are provided:

Stage 1 Gain	Description
3	Sets Stage 1 amplifier gain 3 and bypasses Stage 2.
4	Sets Stage 1 amplifier gain 4 and bypasses Stage 2.
15	Sets Stage 1 amplifier gain 3 and uses Stage 2 amplifier with gain 5.
20 (Default)	Sets Stage 1 amplifier gain 4 and uses Stage 2 amplifier with gain 5.

CSA Bandwidth

The Bandwidth parameter controls the capacitance load at the output of the Stage 1 amplifier and provides bandwidth adjustment capability, allowing tradeoffs in bandwidth, time delay, and power supply rejection ratio. The bandwidth can be selected from one of the following values:

Bandwidth	Description
Highest (Default)	Highest bandwidth, no capacitance added to Stage 1 output.
MediumHigh	Medium high bandwidth.
MediumLow	Medium low bandwidth.
Lowest	Lowest bandwidth, most capacitance added.

MOD ClockScaler

The MOD ClockScaler parameter allows the input clock (48MHz or 24MHz) to be scaled down by a factor of MOD ClockScaler. Allowed values are from 1 to 256.

MOD Period

This parameter sets the period of the counter. Allowed values are between 0 and 2^{16}–1. The period is loaded into the Period register. The value can be later modified using the FLBUCK_SetPeriod() API function.

MOD DimmingResolution

The MOD DimmingResolution parameter contains a value from 2 to 16. This parameter defines the pseudo-random counter period.

Equation 12

$$Period = 2^{DimmingResolution} - 1$$

MOD PulseWidth

This parameter sets the pulse width of the MOD block output. Allowed values are between 0 and 2^{16}–1. The value can be modified using the API.

MOD SignalDensity

The SignalDensity parameter is a 16-bit value that configures the signal density of the MOD block. Enter a value from 0 to 65535. For the DMM mode the lower nibble of the LSB (lower 4 bits) of the 16-bit value sets the dither signal density and the remaining 12 bits determine a MOD block signal density. At lower signal densities, the dither signal becomes significant and hence the DSM resolution becomes important in determining the flicker frequency.

MOD ModResolution

The ModResolution parameter is a 1-, 2-, 3-, or 4-bit value for the resolution of the DSM Modulator.

MOD InterruptType

This parameter sets the interrupt trigger type. The interrupt can be set so that it triggers on the rising edge of the output signal or on the terminal count of the Count register. The Terminal Count Low Point and Terminal Count High Point parameter choices are not available unless center alignment mode is chosen in the Alignment parameter.

Parameter	Description
Compare True	CPU interrupt triggers at the edge of the output.
Terminal Count	CPU interrupt triggers at the end of the period (valid for right and left alignment only).
Terminal Count Low Point	CPU interrupt triggers at the lowest point of the period (valid for center alignment only).
Terminal Count High Point	CPU interrupt triggers at the highest point of the period (valid for center alignment only).

MOD PhaseShift

This parameter configures the phase shift of the pulse. Enter a value from 0 to 65535. PhaseShift allows staggering of the MOD block phase in the left and right alignment configurations. SyncMode must be enabled to use PhaseShift.

MOD Frequency Compensation

Enables or disables software frequency compensation.

MOD CompareType

This parameter sets the compare function type “less than” or “less than or equal.”

Parameter	Description
Less Than	MOD block output goes high when counter value is less than pulse width value.
Less Than or Equal	MOD block output goes high when counter value is less than or equal to pulse width value.

MOD Alignment

The following options are provided.

Parameter	Description
Left	Left alignment MOD block output signal to period clock.
Center	Center alignment MOD block output signal to period clock.
Right	Right alignment MOD block output signal to period clock.

MOD SyncMode

This parameter sets the individual MOD block modulators to be synchronous with other MOD blocks to allow for synchronization of the independent modulated signals in order to control the phases with respect to one another. In this mode, all of the MOD block modulators are required to be configured with the same frequency and period. The following options are provided:

Parameter	Description
Disable	Disables synchronization of the independent modulated signals.
Enable	Enables synchronization of the independent modulated signals.

MOD LowISR

This parameter enables/disables the MOD block to generate a low priority interrupt.

MOD HighISR

This parameter enables/disables the MOD block to generate a high priority interrupt.

HCT RefHigh

Configures the high reference of the DAC output voltage. The DAC reference dictates the output of the hysteretic controller block and it is important to choose the correct value for this parameter. Use the following formulas:

Equation 13

$$PeakCurrent = Current + \frac{Ripple \times Current}{2}$$

Equation 14

$$SenseVoltage = PeakCurrent \times SenseRegister$$

Equation 15

$$DACVoltage = SenseVoltage \times CSAGain$$

Equation 16

$$DACValue = \frac{DACVoltage \times 255}{VoltageRange}$$

In the last formula VoltageRange is the value of HCT DACVoltageRange parameter (1.3V or 2.6V).

Consider an example LED application intended to drive 350 mA of constant current through the LEDs using Floating Load Buck topology. Assume this circuit uses an external sense resistor of 0.22Ω, the internal current sense amplifier gain is set to 20, the desired ripple is 10% with 1 MHz switch frequency, and the DAC is set in the 2.6V range. For an ideal selection of external components, the RefHigh DAC value is computed as:

Equation 17

$$PeakCurrent(mA) = 350 + \frac{0.1 \times 350}{2} = 367.5$$

Equation 18

$$SenseVoltage(mV) = 367.5 \times 0.22 = 80.85$$

Equation 19

$$DACVoltage(V) = 80.85 \times 20 = 1.617$$

Equation 20

$$DACValue \text{ to be loaded (approximate HEX)} = \frac{1.617 \times 255}{2.6} = 9C$$

Actual voltage on the DAC output depends on this parameter and the DACVoltageRange parameter value.

RefLow

Configures low reference DAC output voltage. The DAC reference dictates the output of the hysteretic controller block and it is important to choose the correct value for this parameter. Use the following formulas:

Equation 21

$$ValleyCurrent = Current - \frac{Ripple \times Current}{2}$$

Equation 22

$$SenseVoltage = ValleyCurrent \times SenseRegister$$

Equation 23

$$DACVoltage = SenseVoltage \times CSAGain$$

Equation 24

$$DACValue = \frac{DACVoltage \times 255}{VoltageRange}$$

For example, a LED application intended to drive 350 mA of constant current through the LEDs using Floating Load Buck topology. Assume this circuit uses an external sense resistor of 0.22Ω, the internal current sense amplifier gain is set to 20, the desired ripple is 10% with 1 MHz switch frequency, and the DAC is set in the 2.6V range. For an ideal selection of external components, the RefHigh DAC value is computed as follows:

Equation 25

$$ValleyCurrent(mA) = 350 - \frac{0.1 \times 350}{2} = 332.5$$

Equation 26

$$SenseVoltage(mV) = 332.5 \times 0.22 = 73.15$$

Equation 27

$$DACVoltage(V) = 73.15 \times 20 = 1.463$$

Equation 28

$$DACValue \text{ to be loaded (approximate HEX)} = \frac{1.463 \times 255}{2.6} = 8F$$

The actual voltage on the DAC output depends on this parameter and the DACVoltageRange parameter value.

HCT DACVoltageRange

Selects the voltage range of the reference DACs. The choices are 1.3V and 2.6V.

HCT Feedback Input

Defines the HCT Feedback Input connection. The choices are one of the CSA block outputs or FN0 pins. The specific CSA block or FN0x pin available depends on placement of the user module.

Parameter	Description
CSA	HCT block takes feedback input from CSA block
FN_0_x	HCT block takes feedback input from FN_0_x port

HCT DimInput

Selects the HCT block's dimming input connection. It is possible to connect to one of the MOD block outputs or to one of FN0 pins. A MOD block can provide a PRISM, PWM, or DMM modulated signal as the DIM input.

HCT TripInput

Selects the TRIP input connection to the HYSTCTRL block. It is possible to connect one of the hardware comparators, one of the FN0x pins or any of Vgnd (Vgnd0 or Vgnd1) as input.

HCT TimerDelay

Sets the monoshot timer delay for both on and off timers. Possible choices are no delay, 10-25 ns delay, 20-50 ns delay, or 40-100 ns for both on and off timers.

HCT GateDriver

Configures the operation of the gate drivers. Possible choices are disable, internal driver enabled or external driver enabled.

HCT GateDriverStrength

Configures the strength of the gate drivers. Possible choices are default, 75% of default, 50% of default, and 25% of default.

Application Programming Interface

The Application Programming Interface (API) routines are provided as part of the user module to allow the designer to deal with the module at a higher level. This section specifies the interface to each function together with related constants provided by the include files.

Each time a user module is placed, it is assigned an instance name. By default, PSoC Designer assigns the FLBUCK_1 to the first instance of this user module in a given project. It can be changed to any unique value that follows the syntactic rules for identifiers. The assigned instance name becomes the prefix of every global function name, variable and constant symbol. In the following descriptions the instance name has been shortened to FLBUCK for simplicity.

Note

** In this, as in all user module APIs, the values of the A and X register may be altered by calling an API function. It is the responsibility of the calling function to preserve the values of A and X before the call if those values are required after the call. This "registers are volatile" policy was selected for efficiency reasons and has been in force since version 1.0 of PSoC Designer. The C compiler automatically takes care of this requirement. Assembly language programmers must ensure their code observes the policy, too. Though some user module API function may leave A and X unchanged, there is no guarantee they may do so in the future.

For Large Memory Model devices, it is also the caller's responsibility to preserve any value in the CUR_PP, IDX_PP, MVR_PP, and MVW_PP registers. Even though some of these registers may not be modified now, there is no guarantee that will remain the case in future releases.

The following table represents summary of the API function grouped by User Module configuration types. Current sense amplifier block, modulation block and hysteretic controller block names are abbreviated to CSA, MOD and HCT correspondingly.

API function	PWM Mode	DMM Mode	PRISM Mode
void FLBUCK_Start(void)	●	●	●
void FLBUCK_Stop(void)	●	●	●
CSA block			
void FLBUCK_SetGain(BYTE bGain)	●	●	●
void FLBUCK_SetBandwidth(BYTE bBandwidth)	●	●	●
MOD block			
void FLBUCK_EnableHPInt(void)	●	●	
void FLBUCK_EnableLPInt(void)	●	●	
void FLBUCK_DisableHPInt(void)	●	●	
void FLBUCK_DisableLPInt(void)	●	●	
void FLBUCK_EnableHPIntGlobal(void)	●	●	
void FLBUCK_EnableLPIntGlobal(void)	●	●	
void FLBUCK_DisableHPIntGlobal(void)	●	●	
void FLBUCK_DisableLPIntGlobal(void)	●	●	
void FLBUCK_EnableSyncMode(void)	●	●	

API function	PWM Mode	DMM Mode	PRISM Mode
void FLBUCK_DisableSyncMode(void)	●	●	
void FLBUCK_EnableSyncGlobal(void)	●	●	
void FLBUCK_DisableSyncGlobal(void)	●	●	
void FLBUCK_SetAsSyncMaster(void)	●	●	
void FLBUCK_SetClockScaler(BYTE bClockScaler)	●	●	●
void FLBUCK_SetAlignment(BYTE bAlignment)	●	●	
void FLBUCK_SetPhaseShift(WORD wPhaseShift)	●	●	
void FLBUCK_SetPeriod(WORD wPeriod)	●	●	
void FLBUCK_SetPulseWidth(WORD wPulseWidth)	●		
WORD FLBUCK_wGetPulseWidth(void)	●		
void FLBUCK_SetSignalDensity(WORD wSignalDensity)		●	●
void FLBUCK_SetDimmingResolution(BYTE bResolution)			●
void FLBUCK_EnableFrequencyComp(void)			●
void FLBUCK_DisableFrequencyComp(void)			●
void FLBUCK_SetModResolution(BYTE bModResolution)		●	
HCT block			
void FLBUCK_SetDACVoltageRange(BYTE bVoltageRange)	●	●	●

API function	PWM Mode	DMM Mode	PRISM Mode
void FLBUCK_SetRefHigh(BYTE bData)	●	●	●
void FLBUCK_SetRefLow(BYTE bData)	●	●	●
void FLBUCK_EnableHystTripping(void)	●	●	●
void FLBUCK_DisableHystTripping(void)	●	●	●
void FLBUCK_SetGateDriver(BYTE bSelect)	●	●	●
void FLBUCK_SetGateDriverStrength(BYTE bStrength)	●	●	●
void FLBUCK_SetTimerDelay(BYTE bDelay)	●	●	●

FLBUCK_Start

Description:

Starts the FLBUCK operation.

C Prototype:

```
void FLBUCK_Start(void);
```

Assembler:

```
lcall FLBUCK_Start
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

FLBUCK_Stop

Description:

Stops the FLBUCK operation.

C Prototype:

```
void FLBUCK_Stop();
```

Assembler:

```
lcall FLBUCK_Stop
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

FLBUCK_SetGain

Description:

Sets the current sense amplifier gain. Controls both Stage 1 gain and Stage 2 bypass.

C Prototype:

```
void FLBUCK_SetGain(BYTE bGain);
```

Assembler:

```
mov   A, bGain
lcall  FLBUCK_SetGain
```

Parameters:

bGain – the gain value. Symbolic names are provided in C and assembly. Their associated values are given in the following table:

Symbolic Name	Value	Description
FLBUCK_GAIN_3	0x0A	Sets Stage 1 amplifier gain 3 and bypasses Stage 2.
FLBUCK_GAIN_4	0x08	Sets Stage 1 amplifier gain 4 and bypasses Stage 2.
FLBUCK_GAIN_15	0x02	Sets Stage 1 amplifier gain 3 and uses Stage 2 amplifier with gain 5.
FLBUCK_GAIN_20	0x00	Sets Stage 1 amplifier gain 4 and uses Stage 2 amplifier with gain 5.

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

FLBUCK_SetBandwidth

Description:

Controls the capacitance load at the output of the Stage 1 amplifier and provides bandwidth adjustment capability, allowing tradeoffs in bandwidth, time delay, and power supply rejection ratio.

C Prototype:

```
void FLBUCK_SetBandwidth(BYTE bBandwidth);
```

Assembler:

```
mov   A, bBandwidth
lcall  FLBUCK_SetBandwidth
```

Parameters:

bBandwidth – the bandwidth value. Symbolic names are provided in C and assembly. Their associated values are given in the following table:

Symbolic Name	Value	Description
FLBUCK_BANDWIDTH_HIGHEST	0x00	Highest bandwidth, no capacitance added.
FLBUCK_BANDWIDTH_MEDIUM_HIGH	0x10	Medium high bandwidth.
FLBUCK_BANDWIDTH_MEDIUM_LOW	0x20	Medium low bandwidth.
FLBUCK_BANDWIDTH_LOWEST	0x30	Lowest bandwidth, most capacitance added.

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

FLBUCK_EnableHPInt

Description:

Enables the MOD block to generate a high priority interrupt. This high priority interrupt is shared between all four PSoC MOD blocks.

C Prototype:

```
void FLBUCK_EnableHPInt(void);
```

Assembler:

```
lcall  FLBUCK_EnableHPInt
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

FLBUCK_DisableHPInt

Description:

Disables the MOD block to generate a high priority interrupt. This high priority interrupt is shared between all four PSoC MOD blocks.

C Prototype:

```
void  FLBUCK_DisableHPInt(void);
```

Assembler:

```
lcall  FLBUCK_DisableHPInt
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

FLBUCK_EnableLPInt

Description:

Enables the MOD block to generate a low priority interrupt. This low priority interrupt is shared between all four PSoC MOD blocks.

C Prototype:

```
void FLBUCK_EnableLPInt(void);
```

Assembler:

```
lcall  FLBUCK_EnableLPInt
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

FLBUCK_DisableLPInt

Description:

Disables the MOD block to generate a low priority interrupt. The low priority interrupt is shared between all four PSoC MOD blocks.

C Prototype:

```
void FLBUCK_DisableLPInt(void);
```

Assembler:

```
lcall  FLBUCK_DisableLPInt
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

FLBUCK_EnableHPIntGlobal

Description:

Enables the high priority MOD interrupt operation. The high priority MOD interrupt is shared between all four PSoC MOD blocks.

C Prototype:

```
void FLBUCK_EnableHPIntGlobal(void);
```

Assembler:

```
lcall  FLBUCK_EnableHPIntGlobal
```

Parameters:

None

Return Value:

None

Side Effects:

Calling this function takes control over the interrupt that is related to all four PSoC MOD blocks.

See Note ** at the beginning of the API section.

FLBUCK_DisableHPIntGlobal

Description:

Disables the high priority MOD interrupt operation. The high priority MOD interrupt is shared between all four PSoC MOD blocks.

C Prototype:

```
void FLBUCK_DisableHPIntGlobal(void);
```

Assembler:

```
lcall  FLBUCK_DisableHPIntGlobal
```

Parameters:

None

Return Value:

None

Side Effects:

Calling this function takes control over the interrupt that is related to all four PSoC MOD blocks.

See Note ** at the beginning of the API section.

FLBUCK_EnableLPIntGlobal

Description:

Enables the low priority MOD interrupt operation. The low priority interrupt is shared between all four PSoC MOD blocks.

C Prototype:

```
void FLBUCK_EnableLPIntGlobal(void);
```

Assembler:

```
lcall  FLBUCK_EnableLPIntGlobal
```

Parameters:

None

Return Value:

None

Side Effects:

Calling this function takes control over the interrupt that is related to all four PSoC MOD blocks.

See Note ** at the beginning of the API section.

FLBUCK_DisableLPIntGlobal

Description:

Disables the low priority MOD interrupt operation. The low priority MOD interrupt is shared between all four PSoC MOD blocks.

C Prototype:

```
void FLBUCK_DisableLPIntGlobal(void);
```

Assembler:

```
lcall FLBUCK_DisableLPIntGlobal
```

Parameters:

None

Return Value:

None

Side Effects:

Calling this function takes control over the interrupt that is related to all four PSoC MOD blocks. See Note ** at the beginning of the API section.

FLBUCK_EnableSyncMode

Description:

Enables the MOD to participate in SyncMode operation.

C Prototype:

```
void FLBUCK_EnableSyncMode(void);
```

Assembler:

```
lcall FLBUCK_EnableSyncMode
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

FLBUCK_DisableSyncMode

Description:

Disables the MOD from participation in SyncMode operation.

C Prototype:

```
void FLBUCK_DisableSyncMode(void);
```

Assembler:

```
lcall FLBUCK_DisableSyncMode
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

FLBUCK_EnableSyncGlobal

Description:

Enables global synchronous operation. Note that changing the period and phase shift is allowed only when global synchronization is disabled.

C Prototype:

```
void FLBUCK_EnableSyncGlobal(void);
```

Assembler:

```
lcall FLBUCK_EnableSyncGlobal
```

Parameters:

None

Return Value:

None

Side Effects:

This function affects settings shared b all four PSoC MOD blocks.

See Note ** at the beginning of the API section.

FLBUCK_DisableSyncGlobal

Description:

Disables global synchronous operation. Note that changing the period and phase shift is allowed only when global synchronization is disabled.

C Prototype:

```
void FLBUCK_DisableSyncGlobal(void);
```

Assembler:

```
lcall FLBUCK_DisableSyncGlobal
```

Parameters:

None

Return Value:

None

Side Effects:

This function affects settings shared by all four PSoC MOD blocks.

See Note ** at the beginning of the API section.

FLBUCK_SetAsSyncMaster

Description:

Sets the MOD block to be master. For correct SyncMode operation, two or more MOD blocks must have the same period and clock scaler.

C Prototype:

```
void FLBUCK_SetAsSyncMaster(void);
```

Assembler:

```
lcall  FLBUCK_SetAsSyncMaster
```

Parameters:

None

Return Value:

None

Side Effects:

This function affects settings shared by all four PSoC MOD blocks.

See Note ** at the beginning of the API section.

FLBUCK_SetClockScaler

Description:

Writes the Clock Scaler register with an 8-bit value. The input clock (48 MHz or 24 MHz) is scaled down by a factor of ClockScaler.

C Prototype:

```
void FLBUCK_SetClockScaler(BYTE bClockScaler);
```

Assembler:

```
mov  A, [bClockScaler]
lcall FLBUCK_SetClockScaler
```

Parameters:

bClockScaler – a value from 1 to 256.

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

FLBUCK_SetAlignment

Description:

Selects an alignment value.

C Prototype:

```
void FLBUCK_SetAlignment(BYTE bAlignment);
```

Assembler:

```
mov  A, [bAlignment]
lcall FLBUCK_SetAlignment
```

Parameters:

bAlignment – this parameter lets the user select MOD output waveform alignment. The following options are provided:

Parameter	Value	Description
FLBUCK_LEFT_ALIGNMENT	0x00	Left alignment to period clock.
FLBUCK_CENTRE_ALIGNMENT	0x04	Center alignment (with even period and even duty cycles) to period clock.
FLBUCK_RIGHT_ALIGNMENT	0x08	Right alignment to period clock.

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

FLBUCK_SetPhaseShift

Description:

This API configures the phase of the pulse. This phase control allows for staggering of the PWM or DMM phase in the left and right aligned configurations.

C Prototype:

```
void FLBUCK_SetPhaseShift(WORD wPhaseShift);
```

Assembler:

```
mov   X, [wPhaseShift]
mov   A, [wPhaseShift+1]
lcall  FLBUCK_SetPhaseShift
```

Parameters:

wPhaseShift – allowed values for this field are between zero and 2^{16}–1.

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

FLBUCK_SetPeriod

Description:

Writes the Period register with the period value. The period value be transferred from the Period register to the Counter register immediately, if the MOD is stopped.

C Prototype:

```
void FLBUCK_SetPeriod(WORD wPeriod);
```

Assembler:

```
mov   X, [wPeriod]
mov   A, [wPeriod+1]
lcall  FLBUCK_SetPeriod
```

Parameters:

wPeriod – period value is a value from 0 to 2^{16}–1.

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

FLBUCK_SetPulseWidth

Description:

Writes the Pulse Width register with the pulse width value. Writing the Pulse Width register while the counter is active changes the duty cycle of the output. This may cause the output to glitch or change inadvertently.

C Prototype:

```
void FLBUCK_SetPulseWidth(WORD wPulseWidth);
```

Assembler:

```
mov X, [wPulseWidth]
mov A, [wPulseWidth+1]
```

```
lcall  FLBUCK_SetPulseWidth
```

Parameters:

wPulseWidth – pulse width value is the value from 0 to the period value.

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

FLBUCK_ wGetPulseWidth

Description:

Returns current Pulse Width value.

C Prototype:

```
WORD FLBUCK_wGetPulseWidth(void);
```

Assembler:

```
lcall  FLBUCK_wGetPulseWidth
mov [wPulseWidth + 0], X
mov [wPulseWidth + 1], A
```

Parameters:

None

Return Value:

16-bit value that was read from PulseWidth register.

Side Effects:

See Note ** at the beginning of the API section.

FLBUCK_SetSignalDensity

Description:

Writes the Signal Density register with the signal density value. If Frequency Compensation mode is enabled (for PRISM mode only), this API also adjusts the Frequency Scaler block to keep a stable output frequency.

C Prototype:

```
void FLBUCK_SetSignalDensity(WORD wSignalDensity);
```

Assembler:

```
mov  X, [wSignalDensity]
mov  A, [wSignalDensity + 1]
lcall FLBUCK_SetSignalDensity
```

Parameters:

wSignalDensity – 16-bit signal density value. MSB is passed in the X register and LSB is passed in the Accumulator

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

FLBUCK_SetDimmingResolution

Description:

Initializes the pseudo-random counter Polynomial Register with a polynomial that matches the desired resolution.

C Prototype:

```
void FLBUCK_SetDimmingResolution(BYTE bResolution);
```

Assembler:

```
mov  A, bResolution
lcall FLBUCK_SetDimmingResolution
```

Parameters:

bResolution – the resolution parameter contains a value from 2 to 16 (which be presented by 1 byte). This parameter defines the pseudo-random counter period as 2^n–1 where n is resolution.

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

FLBUCK_EnableFrequencyComp

Description:

Enables the software frequency compensation.

C Prototype:

```
void FLBUCK_EnableFrequencyComp(void);
```

Assembler:

```
lcall FLBUCK_EnableFrequencyComp
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

FLBUCK_DisableFrequencyComp

Description:

Disables the software frequency compensation.

C Prototype:

```
void FLBUCK_DisableFrequencyComp(void);
```

Assembler:

```
lcall  FLBUCK_DisableFrequencyComp
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

FLBUCK_SetModResolution

Description:

Sets the resolution of the modulator.

C Prototype:

```
void FLBUCK_SetModResolution(BYTE bModResolution);
```

Assembler:

```
mov  A, [bModResolution]
lcall FLBUCK_SetModResolution
```

Parameters:

bModResolution – one of the following four values is provided:

Parameter	Value	Modulator Resolution
FLBUCK_RESOLUTION_1BIT	0x30	1 bit
FLBUCK_RESOLUTION_2BIT	0x20	2 bit
FLBUCK_RESOLUTION_3BIT	0x10	3 bit
FLBUCK_RESOLUTION_4BIT	0x00	4 bit

The resolution value is passed in the A register.

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

FLBUCK_SetDACVoltageRange

Description:

Sets the range of hysteretic DAC output voltage. This single parameter sets the range for both reference DACs.

C Prototype:

```
void FLBUCK_SetDACVoltageRange(BYTE bVoltageRange);
```

Assembler:

```
mov  A, bVoltageRange
lcall FLBUCK_SetDACVoltageRange
```

Parameters:

bVoltageRange – indicates the output range of the reference DACs. Symbolic names provided in C and assembly, and their associated values, are given in the following table:

Symbolic Name	Value	Description
FLBUCK_DAC_1_3V	0x02	Sets the reference DAC voltage range from 0V to 1.3V.
FLBUCK_DAC_2_6V	0x00	Sets the reference DAC voltage range from 0V to 2.6V.

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

FLBUCK_SetRefHigh

Description:

Writes an 8-bit data code to the high reference DAC data register. This changes the high reference DAC output voltage setup.

C Prototype:

```
void FLBUCK_SetRefHigh(BYTE bData);
```

Assembler:

```
mov   A, bData
lcall  FLBUCK_SetRefHigh
```

Parameters:

bData – defines the 8-bit data code to be loaded in the high reference DAC.

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

FLBUCK_SetRefLow

Description:

Writes an 8-bit data code to low reference DAC data register. This changes the low reference DAC output voltage setup.

C Prototype:

```
void FLBUCK_SetRefLow(BYTE bData);
```

Assembler:

```
mov   A, bData
lcall  FLBUCK_SetRefLow
```

Parameters:

bData – defines the 8-bit data code to be loaded in the low reference DAC.

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

FLBUCK_EnableHystTripping

Description:

Enables the hysteretic tripping operation. The trip function is generated from the comparator bank through a digital mux. It notifies the hysteretic controller that an over current condition has occurred at the output.

C Prototype:

```
void FLBUCK_EnableHystTripping(void);
```

Assembler:

```
lcall  FLBUCK_EnableHystTripping
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

FLBUCK_DisableHystTripping

Description:

Disables the hysteretic tripping operation.

C Prototype:

```
void FLBUCK_DisableHystTripping(void);
```

Assembler:

```
lcall  FLBUCK_DisableHystTripping
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

FLBUCK_ResetTrip

Description:

Resumes Hysteretic operation after Trip condition.

C Prototype:

```
void FLBUCK_ResetTrip(void);
```

Assembler:

```
lcall  FLBUCK_ResetTrip
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

FLBUCK_SetGateDriver

Description:

Configures the operation of the internal and external gate drivers.

C Prototype:

```
void FLBUCK_SetGateDriver(BYTE bSelect);
```

Assembler:

```
mov   A, bSelect
lcall  FLBUCK_SetGateDriver
```

Parameters:

bSelect – specifies the enabled gate driver. Symbolic names provided in C and assembly, and their associated values, are given in the following table:

Symbolic Name	Value	Description
FLBUCK_GDSEL_NONE	0x00	Disable both drivers.
FLBUCK_GDSEL_EXTERNAL	0x01	Enable the external driver only.
FLBUCK_GDSEL_INTERNAL	0x02	Enable the internal driver only.

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

FLBUCK_SetGateDriverStrength

Description:

Sets the drive strength of the gate driver.

C Prototype:

```
void FLBUCK_SetGateDriverStrength(BYTE bStrength);
```

Assembler:

```
mov   A, bStrength
lcall  FLBUCK_SetGateDriverStrength
```

Parameters:

bStrength – specifies the gate driver strength. Symbolic names provided in C and assembly, and their associated values, are given in the following table:

Symbolic Name	Value	Description
FLBUCK_STRENGTH_DEFAULT	0x00	Default drive strength.
FLBUCK_STRENGTH_75PC	0x04	75% of the default strength.
FLBUCK_STRENGTH_50PC	0x08	50% of the default strength.
FLBUCK_STRENGTH_25PC	0x0C	25% of the default strength.

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

FLBUCK_SetTimerDelay

Description:

Sets the monoshot timer delay for both the on and off timers.

C Prototype:

```
void FLBUCK_SetTimerDelay(BYTE bDelay);
```

Assembler:

```
mov   A, bDelay
lcall  FLBUCK_SetTimerDelay
```

Parameters:

bDelay – specifies the timer delay. Symbolic names provided in C and assembly, and their associated values, are given in the following table:

Symbolic Name	Value	Description
FLBUCK_DELAY_OFF	0x0C	No delay from either timer.
FLBUCK_DELAY_10_25NS	0x00	10 to 25 ns timer delay for both the on and off timers.
FLBUCK_DELAY_20_50NS	0x04	20 to 50 ns timer delay for both the on and off timers.
FLBUCK_DELAY_40_100NS	0x08	40 to100 ns timer delay for both the on and off timers.

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

Sample Firmware Source Code

The C code illustrated here shows you how to use the FLBUCK User Module.

```
FLBUCK_Start();
FLBUCK_SetDACVoltageRange(FLBUCK_DAC_2_6V);
FLBUCK_SetRefHigh(105);
FLBUCK_SetRefLow(95);
```

The same code in assembly is:

```
call  FLBUCK_Start
mov   A, FLBUCK_DAC_2_6V
call  FLBUCK_SetDACVoltageRange
mov   A, 105
call  FLBUCK_SetRefHigh
mov   A, 95
call  FLBUCK_SetRefLow
```

Configuration Registers

The FLBUCK User Module is personalized and parameterized through the registers. The following tables shows the register values as constants and the parameters as named bit fields with brief descriptions.

Table 1. HCT_VDAC_CONTROL_REG

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	Mode	Enable

Enable enables or disables the VDAC. Disabling powers down the VDAC and all of its output references go to 0V. It is modified by calling FLBUCK_Start or FLBUCK_Stop.

Mode sets the VDAC output range and step size. The initial value of this bit is set with the value of the DACVoltageRange parameter in the Device Editor. The value can be changed at runtime with the FLBUCK_SetDACVoltageRange API.

Table 2. HCT_VDAC_DATA0_REG

Bit	7	6	5	4	3	2	1	0
Value	RefHigh							

RefHigh determines the reference high DAC output voltage. The initial value of this bit is set with the value of the RefHigh parameter in the Device Editor. The value can be changed at runtime with the FLBUCK_SetRefHigh API.

Table 3. HCT_VDAC_DATA1_REG

Bit	7	6	5	4	3	2	1	0
Value	RefLow							

RefLow determines the reference low DAC output voltage. The initial value of this bit is set with the value of the RefLow parameter in the Device Editor. The value can be changed at runtime with the FLBUCK_SetRefLow API.

Table 4. HCT_HYST_CONTROL_REG

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	TimerDelay		Trip	Enable

Enable enables or disables the user module. It is modified by calling FLBUCK_Start or FLBUCK_Stop.

Trip determines the hysteretic controller tripping ability and can be changed at runtime with the FLBUCK_EnableHystTripping API.

TimerDelay determines the monoshot timer delay for both on and off timers. The initial value of this bit is set with the value of the TimerDelay parameter in the Device Editor. The value can be changed at runtime with the FLBUCK_SetTimerDelay API.

Table 5. HCT_CMP_CONTROL_REG

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	Enable	0	0	0	Enable

Enable enables or disables the comparator block (even and odd) in the hysteretic channel. It is modified by calling FLBUCK_Start or FLBUCK_Stop.

Table 6. HCT_GATE_CONTROL_REG

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	GateDriverStrength		GateDriver	

GateDriver selects the active FET gate driver. The initial value of this bit is set with the value of the GateDriver parameter in the Device Editor. The value can be changed at runtime with the FLBUCK_SetGateDriver API.

GateDriverStrength selects the FET gate driver strength. The initial value of this bit is set with the value of the GateDriverStrength parameter in the Device Editor. The value can be changed at runtime with the FLBUCK_SetGateDriverStrength API.

Table 7. CSA_CONTROL_REG

Bit	7	6	5	4	3	2	1	0
Value	0	0	Bandwidth		Gain			Enable

The Enable bit is modified by calling the FLBUCK_Start or FLBUCK_Stop API routine.

The Gain bits determine the Stage 1 gain and control the Stage 2 amplifier bypass. The value of these bits is determined by the choice made for the parameter of the same name under user module parameters in the Device Editor. The value can also be changed by calling the FLBUCK_SetGain API.

The Bandwidth bits determine the CSA bandwidth. The value of these bits is determined by the choice made for the parameter of the same name under user module parameters in the Device Editor. The value can also be changed by calling the FLBUCK_SetBandwidth API.

Table 8. MOD_PCF_REG

Bit	7	6	5	4	3	2	1	0
Value	Programmable Clock Frequency Scaler							

Programmable Clock Frequency Scaler determines the clock scaler for the MOD block. The value of this register is determined by the choice made for the ClockScaler parameter in the user module parameters of the Device Editor. This value can also be changed by the FLBUCK_SetClockScaler() API..

Table 9. MOD_PDH_REG (MSB), MOD_PDL_REG (LSB)

Bit	7	6	5	4	3	2	1	0
LSB	Period/Polynomial Register (LSB)							
MSB	Period/Polynomial Register (MSB)							

- For PWM mode Period determines the period value for the MOD block. The value of this register is determined by the choice made for the Period parameter in the user module parameters of the Device Editor. This value can also be changed by the FLBUCK_SetPeriod API.
- For DMM mode Period determines the period value for the MOD block. This value is 12-bit wide and therefore 4 most significant bits in MSB register are not used. The value of this register is determined

by the choice made for the Period parameter in the user module parameters of the Device Editor. This value can also be changed by the FLBUCK_SetPeriod API.

- For PRISM mode Polynomial determines the period value for the MOD block. The value of this register is determined by the choice made for the DimmingResolution parameter in the user module parameters of the Device Editor also value can be changed by FLBUCK_SetDimmingResolution API..
- For PWM mode Pulse Width determines the pulse width value for the MOD block. The value of this register is determined by the choice made for the PulseWidth parameter in the user module parameters of the Device Editor. This value can also be changed by the FLBUCK_SetPulseWidth API.
- For DMM mode Signal Density determines the signal density value for the MOD block. The value of this register is determined by the choice made for the SignalDensity parameter in the user module parameters of the Device Editor. This value can also be changed by the FLBUCK_SetSignalDensity API.
- For PRISM mode Signal Density determines the signal density value for the MOD block. The value of this register is determined by the choice made for the SignalDensity parameter in the user module parameters of the Device Editor. This value can also be changed by the FLBUCK_SetSignalDensity API.

Table 10. MOD_PWH_REG (MSB), MOD_PWL_REG (LSB)

Bit	7	6	5	4	3	2	1	0
LSB	Pulse Width/Signal Density Register (LSB)							
MSB	Pulse Width/Signal Density Register (MSB)							

Table 11. MOD_PCH_REG (MSB), MOD_PCL_REG (LSB)

Bit	7	6	5	4	3	2	1	0
LSB	Phase Shift Register (LSB)							
MSB	Phase Shift Register (MSB)							

Phase Shift determines the phase shift value for the MOD block. The value of this register is determined by the choice made for the PhaseControl parameter in the user module parameters of the Device Editor. This value can also be changed by the FLBUCK_SetPhaseShift API.

Table 12. MOD_PCFG_REG

Bit	7	6	5	4	3	2	1	0
Value	0	IntMode	ModResolution		Align[1:0]		CompType	IntType

IntType and IntMode determine the MOD interrupt type. The value of these bits is determined by the choice made for the InterruptType parameter in the user module parameters of the Device Editor.

ModResolution determines the pulse alignment configuration for MOD block in DMM mode. The value of these bits is determined by the choice made for the parameter with the same name in the user module parameters of the Device Editor. This value can be changed by the FLBUCK_SetModResolution API.

CompType determines the MOD compare type. The value of this bit is determined by the choice made for the CompareType parameter in the user module parameters of the Device Editor.

Align[1:0] determines the MOD pulse alignment configuration. The value of these bits is determined by the choice made for the Alignment parameter in the user module parameters of the Device Editor. This value can also be changed by the FLBUCK_SetAlignment API.

Table 13. MOD_GCFG_REG

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	Enable

Enable turns on the MOD block. The value of this bit can be changed by the FLBUCK_Start and FLBUCK_Stop APIs.

Version History

Version	Originator	Description
1.1	DHA	Added ResetTrip API FLBUCK User Module
1.1.b	DHA	Added wizard help file.

Note PSoC Designer 5.1 introduces a Version History in all user module datasheets. This section documents high level descriptions of the differences between the current and previous user module versions.

Copyright © 2009-2012 Cypress Semiconductor Corporation. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

PSoC Designer™ and Programmable System-on-Chip™ are trademarks and PSoC® is a registered trademark of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.